THE UNIVERSITY
of EDINBURGH

Controlling optical beams in nematic liquid crystals

Bryan Tope

Doctor of Philosophy
University of Edinburgh
April 2017

Declaration

I declare that this thesis was composed by myself and that the work contained therein is my own, except where explicitly stated otherwise in the text.

(*Bryan Tope*)

To my sons Mackenzie and Cameron

Abstract

A major area of research recently has been the study of nonlinear waves in liquid crystals. The availability of commercial liquid crystals and the formation of solitons at mW power levels has meant that experimental research and the need to understand how the solitons are formed and interact has been boosted. The first part of the thesis looks at how two laser beams in a nematic liquid crystal interact. Specifically research has centred on the problem of directing a signal beam to a target area by varying the input angle of the control beam. Different approximate models are developed to describe this phenomena, with the results from these models compared to a full numerical analysis.

The first model developed is called the particle model and is based on the unmodified modulation equations. The results from this model were acceptable when compared with the results obtained from a full numerical analysis. This comparison is indicative that the underlying assumptions of the model did not capture an essential part of interaction between the two laser beams. The second model used to describe the interaction between the two laser beams was based on the law of conservation of momentum in the laser beams. Here the potential between the laser beams was modified to take into account the profile of the beams. The results from this model were in excellent agreement with results from the full numerical analysis, showing the key role potential between the beams plays in the trajectories of the beams.

The interaction between dark solitons was also studied. The model used in this case was based on the modulation equations with a suitable trial function for dark solitons. The results from this model were in excellent agreement with the results from the full numerical analysis. The behaviour of the dark solitons shown by the approximate model and the full numerical analysis showing similar key features.

This thesis sets out the equations describing the interaction of laser beams in liquid crystals. These are the equations used to carry out a full numerical analysis. This analysis is valuable in its own right and is the standard to compare the results obtained from other models but to achieve a deeper understanding of how laser beams interact in liquid crystals approximate models are developed so that the important parameters in each model can be identified. The Lagrangian describing the interaction of laser beams in liquid crystals is used in all the approximate models.The approximate models can then be developed through the use of suitable trial functions that adequately describe how the laser beams interact. The derivation of the equations and how these equations are solved is described for each model. The results from each model are compared to a full numerical analysis with a discussion of how the results compare.

Acknowledgements

First of all, I would like to thank my supervisor Professor Noel Smyth for all his help, guidance and common interests during my time as a PhD student. During this time I have been able to meet many experts in the areas of mathematics, physics and material science from all over the world. They have all shown a patience and interest in my studies and at times I have managed to pick up some of their deep understanding in the nonlinear waves world. I would also like to thank Dr Bruce Worton for his guidance and help in delivering a course at the University of Edinburgh. Doing a PhD has changed me in many ways and the future I face is now very different to the one I faced before I started my PhD. Thanks to you all.

Contents

Chapter 1

Introduction

1.1 Solitons

The story of solitons begins with John Scott Russell who first noticed them in 1834 while riding along the Union canal near Edinburgh. This is how he described the phenomenon in his report to the British Association for the Advancement of Science [1].

I was observing the motion of a boat which was rapidly drawn along a narrow channel by a pair of horses, when the boat suddenly stopped not so the mass of water in the channel which it had put in motion; it accumulated round the prow of the vessel in a state of violent agitation, then suddenly leaving it behind, rolled forward with great velocity, assuming the form of a large solitary elevation, a rounded, smooth and well-defined heap of water, which continued its course along the channel apparently without change of form or diminution of speed. I followed it on horseback, and overtook it still rolling on at a rate of some eight or nine miles an hour , preserving its original figure some thirty feet long and a foot to a foot and a half in height. Its height gradually diminished, and after a chase of one or two miles I lost it in the windings of the channel. Such, in the month of August 1834, was my first chance interview with that singular and beautiful phenomenon which I have called the Wave of Translation.

This observation suggested that permanent waves of hump form could exist, which contradicted what was known about water waves at that time. The great G.G. Stokes stated that the wave observed by Russell could not be permanent. This was because water waves were known to be dispersive. A hump would consist of a number of Fourier modes, which should propagate with their own phase velocity, so that the hump should break up. This understanding of waves was based on the waves being formed by linear processes. It was much later that the theory of nonlinear waves was studied and could be used to explain John Scott Russell's waves. Given this background and the theoretical objections to his conclusions, John Scott Russell knew that he had observed a new phenomena and spent some time trying to gain a better understanding of his "Wave of Translation". He built a wave tank at his home to study his new type of wave. He noted the following key properties of the waves he had observed

- The waves were stable.
- The speed depends on the size of the wave.
- Unlike normal waves they do not merge.

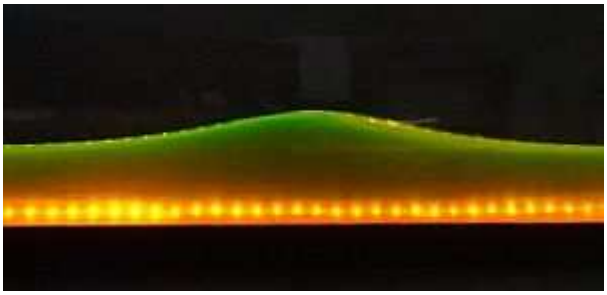

Figure 1.1: Water wave soliton produced in a laboratory wave tank. This is a solution of the KdV equation (1.1). Photograph by Christophe Finot and Kamal Hammani [18]

- If a wave is too big for the depth of the water, it splits into two waves.

The equation describing Russell's "wave of translation" is the Korteweg-de Vries equation

$$\frac{\partial u}{\partial t} + 6u\frac{\partial u}{\partial x} + \frac{\partial^3 u}{\partial x^3} = 0 \tag{1.1}$$

derived in 1895 [3]. This equation has the soliton solution

$$u = a\,\mathrm{sech}^2\sqrt{\frac{a}{2}}(x - 2at). \tag{1.2}$$

The "wave of translation" was a nonlinear wave. This explained why Stokes' objections were not valid, as up to that point in time only linear waves had been studied. The wave did not disperse as the nonlinear response could counter-balance dispersion. The existing theories on water waves could not explain the wave of translation and it was not until the 1870s that an explanation was provided for these waves with the work of Lord Rayleigh [2] and Joseph Boussinesq [3]. This work was extended by Diederick Kortweg and Gustav de Vries in 1895 with the derivation of the KdV equation shown in (1.1). This equation contains the nonlinear component , $6u\frac{\partial u}{\partial x}$, and linear component, $\frac{\partial^3 u}{\partial x^3}$, so that the linear dispersion can be balanced by nonlinear focusing. With these theoretical justifications for the "wave of translation", termed a solitary wave, work in this area ceased. Indeed, in Lamb's text *Hydrodynamics* [4] on the state of fluid mechanics at the beginning of the 20th century, solitary waves are given only a brief mention.

With the advent of modern computers and compilers in the 1960s it became much easier to study nonlinear phenomena and with this advance the significance of solitary waves increased. The word "soliton" was coined when numerical simulations of solitary wave solutions of the KdV equation (1.1) carried out by Kruskal and Zabusky [5] showed the solitary waves did not change form on interaction, the only evidence of the interaction being a phase shift. The waves then acted like bullets or particles, which suggested that there was some sort of underlying linear behaviour to the nonlinear interaction. Figure 1.1 shows an water wave soliton produced in a laboratory. The wave is a solution of the KdV equation (1.1). Note the characteristic shape of the soliton with its $\mathrm{sech}^2(x-t)$ profile and compare with the soliton solution of the KdV equation (1.2).

Shortly after the work of Kruskal and Zabusky, Gardener, Greene, Kruskal and Muira [6] discovered the "inverse scattering transform". This was a general method for

solving some nonlinear partial differential equations. Inverse scattering for the KdV equation reduces the KdV equation to two linear equations [45]. One is Schrödinger's equation from quantum mechanics. The point spectrum of this equation gives the solitons which arise out of the evolution of a (square integrable) initial condition. The other equation is an integral equation, the Marchenko equation, which gives the evolution of the initial condition to this final soliton state. While the point spectrum of Schrödinger's equation is relatively easy to find, there are no known solutions of the Marchenko equation. Hence, the evolution of an initial condition to the final soliton state is not easy to calculate using inverse scattering, even through the solution is given, in principle, by linear equations. With this initial discovery that the KdV equation was integrable (in a Hamiltonian sense), many more nonlinear dispersive wave equations were found to possess an inverse scattering solution, including the nonlinear Schrödinger and Sine-Gordon equations [127].

A consequence of an equation possessing an inverse scattering solution is that its solutions are solitons, that is they interact cleanly with the only trace of the interaction being a phase shift. However, a nonlinear wave equation being exactly integrable is the exception, rather than the rule. Most of the nonlinear wave equations arising in applications are not integrable, including those studied in this thesis. Solitary wave solutions of these equations do not interact cleanly and do undergo changes under such interactions.

The equations solvable by the inverse scattering transform form a class of integrable systems and their solutions behave like solitons when they interact with other solitons. That is they do not change form when they interact with other solitons, the only change being a shift in phase. However the majority of nonlinear dispersive wave equations are not integrable and their solutions often do not meet the exact definition of "solitons" mentioned above. These solutions still retain many of the properties of solitons, so the definition of the term "soliton" was extended to include these solutions as well. The terms "soliton" and "solitary wave" are used interchangeably in the physics literature even though this is not mathematically correct. In this thesis we will adopt this convection and the term "soliton" will be used to describe a localised solution of a nonlinear dispersive partial differential equation for which

- the solution is a wave which retains its basic form over a long time.

- interactions between solitons can be elastic or inelastic. However, solitons emerge from collisions with a similar size and shape or fuse together, creating a larger localised wave with similar shape.

The use of solitons in research has increased exponentially as more and more applications are found where solitons provide a good description of the phenomena involved. Solitons form such a central point in modern scientific research that re-enactments of the original discovery have taken place. In 1995, as part of an international conference on nonlinear waves at Heriot-Watt University, a soliton was created on the same Union canal close to the area where John Scott Russell had first observed a soliton in 1834. The event was witnessed by a group of scientists attending the conference and is illustrated in Figure 1.2. The soliton can clearly be seen ahead of the boat.

As mentioned above the inverse scattering transform was originally used to solve the KdV equation and later its use was extended to solve the nonlinear Schrödinger

Figure 1.2: Recreating a soliton on Union Canal 12 July 1995. The soliton can be seen ahead of the boat and the observers are scientists attending a conference on nonlinear waves at Heriot-Watt University, Edinburgh. Photograph by Prof. Dugald Duncan, Heriot-Watt University [17]

equation [45]

$$i\frac{\partial u}{\partial z} + \frac{1}{2}\nabla_{\perp}^{2}u + g|u|^{2}u = 0. \tag{1.3}$$

The nonlinear Schrödinger equation (1.3) plays a central role in this thesis and in one transverse direction has the soliton solution,

$$u = \operatorname{sech}(x - z)\exp\left(\frac{1}{2}i(z + x)\right). \tag{1.4}$$

A brief look at the history of this equation is helpful in understanding the part it plays in modern physics, and in nonlinear optics in particular. The nonlinear Schrödinger (NLS) equation (1.3) is a prototypical dispersive nonlinear partial differential equation (PDE) that has been derived in many areas of physics and analysed mathematically for over 40 years. Historically, the essence of NLS equations can be found in the early work of Ginzburg and Landau (1950) [7] and Ginzburg (1956) [8] in their study of the macroscopic theory of superconductivity, and also of Ginzburg and Pitaevskii (1958) [9] who subsequently investigated the theory of super-fluidity. Nonetheless, it was not until the works of Chiao et al (1964) [10] and Talanov (1964) [11] that the wider physical importance of NLS equation became evident, especially in connection with the phenomenon of self-focusing in optics and the conditions under which an electromagnetic beam can propagate without spreading in nonlinear media. In the general situation, an optical beam in a dielectric broadens due to diffraction. However,

in materials whose dielectric constant increases with the field intensity, the critical angle for total internal reflection at the beam's boundary can become greater than the angular divergence due to diffraction and, as a consequence, the beam does not spread and can, in some situations, continue to focus into extremely high intensity spots.

Starting from the electromagnetic wave equation in the presence of non-linearities and assuming a linearly polarized wave propagating along the z-axis, after a suitable rescaling of the dependent and independent variables one can derive for the propagation of the electromagnetic field the NLS equation (1.3) in standard non-dimensional form [45]. Here u is the slowly varying complex envelope of the electromagnetic field, z is the propagation variable and $\nabla_\perp$ denotes the Laplacian with respect to the transverse coordinates. For $g = 1$ it is termed the focusing NLS equation and for $g = -1$ it is termed the defocusing NLS equation. This terminology comes from nonlinear optics [27], where the refractive index of the medium increases with the intensity of optical beams in the medium. The increase in refractive index causes the optical beam to focus and balances the diffraction that naturally occurs with light beams. Dyes can be added to the medium so that the opposite effect occurs. The refractive index of the medium decreases with intensity of any optical beam. This causes the optical beam to defocus and when solitons form they are called dark solitons. Solitons for the focusing NLS equation are termed bright solitons and solitons for the de-focusing NLS equations are dark or grey solitons. Bright solitons form a peak against the background radiation, while dark or grey solitons are a dip compared to the background radiation. Beside the fact that NLS systems have direct applications in many physical problems, the importance of the NLS equation is also due to its universal character (cf. Benney and Newell, 1967 [12]). Generically speaking, most weakly nonlinear, dispersive, energy-preserving systems give rise, in an appropriate limit, to the NLS equation. Specifically, the NLS equation provides a "canonical" description for the envelope dynamics of a quasi-monochromatic plane wave propagating in a weakly nonlinear dispersive medium when dissipation can be neglected.

Mathematically, the NLS equation attains broad significance [45] since, in one transverse dimension, it is integrable via the Inverse Scattering Transform (IST, for brevity) – which is a nonlinear Fourier Transform – it admits multi-soliton solutions, it has an infinite number of conserved quantities and it possesses many other interesting properties.

There has been a vast amount of literature involving the NLS equation over the years, but recently there has been additional interest, mainly due to the developments in nonlinear optics and soft condensed matter physics. In the optical context, the experimental developments involving localized pulses in arrays of coupled optical waveguides (e.g. Eisenberg et al, 1998 [13]) have drawn attention to discrete NLS models (where the derivatives of the fields are substituted by appropriate finite differences). Related problems involving NLS equations on a lattice background (cf. Efremidis et al, 2003 [14]) have also generated considerable interest. The vector generalization of the NLS equation has also proved to be particularly valuable from the point of view of nonlinear optics. On the other hand, the experimental realization of Bose-Einstein condensates (BECs) and their mean field modelling by the so-called Gross-Pitaevskii equation (Pethick and Smith, 2002 [15]) which, like optical pulses on a lattice background, is an NLS equation with an external potential, has opened new avenues for the study of NLS-type equations.

Both focusing and defocusing forms of NLS type equations and their solitary wave solutions will be discussed in this thesis in the context of nonlinear optics and nematic liquid crystals. The NLS equation first arose in water wave stability theory as the

focusing NLS equation describes weakly nonlinear modulationally unstable periodic waves [45]. The classic example of unstable nonlinear waves is surface gravity waves, for which the instability is termed the Benjamin-Feir instability [125, 126]. A linearised modulational analysis of the stability of weakly nonlinear surface water waves leads to the prediction of their instability [45]. The addition of weak non-linearity to the modulation analysis leads to the focusing NLS equation [45].

Solitary waves are common in fluid mechanics, optics, atomic physics, biophysics and more [16, 22, 23, 24]. It is impossible to discuss all these applications thoroughly, so only a few will be mentioned. There have been extensive experimental and theoretical investigations about both spatial solitary waves, in which non-linearity balances diffraction, and temporal solitary waves, in which non-linearity balances dispersion. From a mathematical perspective, continuous nonlinear Schrödinger (NLS) equations are among the hallmark models in nonlinear optics, as they describe dispersive envelope waves (via solitary-wave solutions of the NLS) of the electric field of optical beams, and discrete NLS (DNLS) equations can be used to describe the dynamics of pulses in optical waveguide arrays and photorefractive crystals. There have also been numerous studies of light bullets, which are three-dimensional localized pulses in self-focusing media with anomalous group velocity dispersion. The properties of optical solitary waves can be manipulated experimentally through both "dispersion management" and "non-linearity management" [25, 26].

There are many instances of solitons occurring naturally and some of these instances are described below. The more research that is carried out seems to uncover more phenomena where solitons occur naturally and non-linear processes are so ubiquitous that the stability of a natural process is formed around the stability of a soliton.

There is currently substantial interest in the use of optical solitons in fibre optics due to the potentially large increase in speed, or bit rate, which is likely to be obtained by their use. The low losses associated with soliton propagation and the stability of solitons to perturbations encouraged researchers in the communications industry to use solitons in fibre optics. This encouraged further research and initiated the discovery of new types of temporal and spatial solitons in a huge variety of materials. The equation governing the propagation of a pulse in a monomode, polarization-preserving, nonlinear optical fibre in the anomalous group-velocity dispersion regime is the nonlinear Schrödinger (NLS) equation rather than the KdV equation, which can be written in non dimensional form as [32, 33]

$$i\frac{\partial u}{\partial z} + \frac{1}{2}\frac{\partial^2 u}{\partial \tau^2} + |u|^2 u = 0, \tag{1.5}$$

where u is the complex-valued envelope of the pulse. Here z represents physically the normalised spatial variable along the length of the fibre, and τ, rather than being a spatial variable, is the normalized reduced time (i.e., shifted to be in a frame of reference which moves with the group velocity of a pulse). The NLS equation has the well known soliton solution

$$u = \eta \operatorname{sech} \eta(\tau - \tau_0 - Vz)e^{i(\eta^2 - V^2)z/2 + iV\tau + i\theta}, \tag{1.6}$$

where η, τ_0, V and θ are constants [91]. The soliton used in fibre optics is a temporal soliton as opposed to a spatial soliton. Temporal solitons are formed via a balance between nonlinear self-phase modulation and linear dispersion, while spatial solitons rely on balancing nonlinear self-focusing and linear diffractive in the spatial dimension.

Diffractive spreading can be countered by an increase in the refractive index of the medium in the vicinity of the beam, which focuses the beam and causes a waveguiding effect. If the beam itself causes the refractive index change in the medium then the beam traps itself in its own waveguide, a property known as nonlinear local self-focusing, where the term "local" refers to the self-focusing response of the medium being greatest where the beam intensity is high and reducing to zero outside of the near vicinity of the beam.

At very low temperatures, particles in a dilute Bose gas can occupy the same quantum (ground) state, forming a Bose-Einstein condensate (BEC). A BEC is a coherent cloud of atoms which appear as a sharp peak in both position and momentum space [23, 27, 28]. As the gas is cooled, a large fraction of the atoms in the gas condense via a quantum phase transition, which occurs when the wavelengths of individual atoms overlap and behave identically. The macroscopic dynamics of BECs near zero temperature are modelled by an NLS equation known as the Gross-Pitaevskii (GP) equation. BEC solitary waves of numerous types have also been modelled using other models, such as DNLS equations. Because of the similarity of the model equations, many of the solitary wave phenomena that were originally studied in the context of nonlinear optics arise here as well, and the extreme tunability of BECs has been a major boon for both theoretical and experimental studies. Many novel types of solitary wave structures have now been created in BEC laboratories, and research on nonlinear waves in BECs continues to develop at a rapid pace. One of the most important current experimental challenges for work on solitary waves in BECs (and also nonlinear optics) is the creation of stable two dimensional and three dimensional solitary waves in the presence of a cubic self-focusing non-linearity, as such structures must be stabilized in order to prevent them from collapsing (in accord with theoretical predictions) [55].

As we have already seen solitons can occur as water waves. Russell's "wave of translation" was a water wave soliton, and Korteweg and de Vries derived their nonlinear wave equation (1.1) to describe the shallow water waves that Russell had observed. The KdV equation arises in the long wavelength limit, and shallow water solitary waves have been the subject of numerous laboratory experiments. Solitary waves also arise in deep water, as shown by the pioneering work of Vladimir Zakharov who derived an envelope wave description whose limiting case satisfies an NLS equation [29].

Additionally, solitary wave solutions have been constructed in more sophisticated models in fluid dynamics, and there has been a lot of work on many types of solitary waves. An example is the Great Red Spot on Jupiter. This is about $400,000$ km in diameter, reddish-hued, cyclonic weather system which has remained stationary in Jupiter's turbulent atmosphere [65]. This has been stable for over three hundred years and exhibits many soliton features. Laboratory experiments [66] and theoretical analysis [67] support the idea that the Great Red Spot is a soliton solution of the Navier-Stokes equation. More study is still needed to confirm this idea.

Various scientists have also attempted to explain the large and seemingly spontaneous freak waves (or rogue waves) as solitary waves. Additionally, tidal bores have been explained in terms of dispersive shock waves, which consist (in spatial profile) of a leading pattern in the form of a solitary travelling wave and a trailing pattern in the form of a wave train with slowly modulated amplitude that eventually asymptotes to a linear wave.

Figure 1.3 shows solitons occurring in the Lombok strait. The Lombok Strait separates the Indonesian islands Bali and Lombok and is one of the most important straits through which water is exchanged between the Pacific Ocean and the Indian Ocean. Bali is the island left centre of the image and Lombok is the island right centre of the

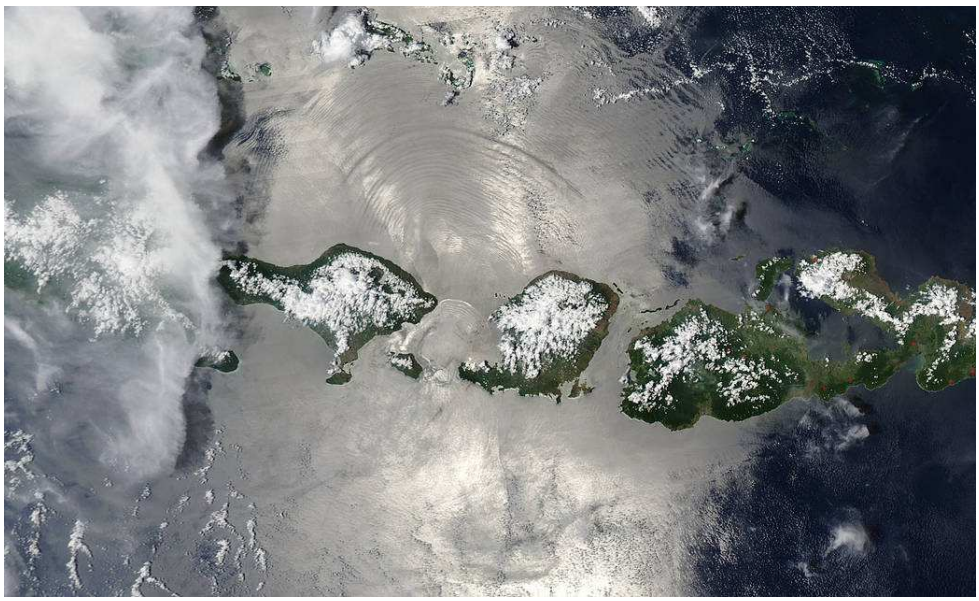

Figure 1.3: Solitons generated in the Lombok strait. This strait is located between the islands of of Bali (centre left) and Lombok (centre right). The strait is an important through-flow between the Pacific ocean (north of the strait) and the Indian ocean (south of the strait). The solitons are generated within the strait and are heading north into the Java sea. Image courtesy of Jeff Schmaltz [19]

image. The Pacific ocean is north of the islands and the Indian ocean is south of the islands. This strait is one of the Indonesian through-flows that plays an important role in the global oceanic circulation.

Transport through the Strait exhibits large seasonal variations due to changes in the atmospheric pressure gradient between the Pacific and the Indian Oceans. As a result, the seasonal currents through the strait are bidirectional. The main topographic features inside the Lombok Strait are an island (Nusapenida) and a sill between this island and the smaller Lombok islands in the southern mouth of the strait. The sill around the island of Nusapenida has a depth of around $200m$, while the depth of the strait either side of the sill is approximately $600 - 800m$. The change in depth of the sill compared to the surrounding strait causes the formation of solitary waves. Figure 1.3 is an ERS SAR (European Remote Sensing Synthetic Aperture Radar) image of this region showing packets of internal waves propagating to the north into the Java Sea where they reach the shallow waters surrounding Pulau-Pulau Kangean.

There have been some attempts to use solitary wave descriptions to describe various biophysical phenomena. One example is the Davydov soliton, which satisfies an equation that was designed to model energy transfer in the hydrogen bonded spines that stabilize protein α-helices [48]. The Davydov soliton represents a state composed of an excitation of amide-I and its associated hydrogen bond distortion. It has been used to describe a local conformational change of the DNA α-helix, and there now exists experimental evidence of such states [48].

A Josephson junction is a nonlinear oscillator consisting of two weakly coupled superconductors that are connected by a non conducting barrier [16]. Such junctions might prove to be important for producing quantum mechanical circuits, such as superconducting quantum interference devices (SQUIDs). The first experimental realization

of an array of such junctions revealed excitations that arose from spatially localized voltage drops at particular junctions as a homogeneous direct bias current traversed an annular array. Solitary waves in "long Josephson junctions", which are much longer than the intrinsic length scale known as the Josephson penetration depth (which is of the order $1 - 1000\mu m$), are known as fluxons because they contain one quantum of magnetic flux.

Numerous interesting nonlinear wave phenomena can occur on the surface of a "continuum" (e.g., fluids, solids, and appropriate granular materials which can often be modelled using continuum descriptions), and some of them admit solitary wave descriptions. Although it can be applied more broadly, the term surface wave is often used to refer to a relatively specific class of examples. These include the pattern forming standing waves called Faraday waves that form, e.g., on the surface of continua housed in vertically vibrated receptacles (similar phenomena have now also been seen in other settings, such as BECs [31]); soliton-like oscillations that switch between peaks and craters and have been demonstrated in vertically vibrated plates of granular materials, viscous fluids, and colloids; and acoustic surface waves, which travel along the surfaces of solid materials.

This background in nonlinear wave theory and solitary waves will now be set in the specific context of nonlinear optical beam evolution in nematic liquid crystals. The next section will look at the properties of nematic liquid crystals and the solitons that are produced. Spatial optical solitons that are produced in nematic liquid crystals are termed "nematicons" [44].

1.2 Nematic Liquid Crystals

Solitons have been observed in many different types of material. The first solitons were observed in water, while more recently solitons have been seen in Bose-Einstein condensates [55]. Another material that has been used extensively in soliton research is nematic liquid crystals. They are widely used due to their easily controlled nonlinear response and support solitons at milliwatt powers.

Liquid crystals display characteristics of liquids and also retain a degree of molecular order similar to crystals. Above a critical temperature all order is lost and the medium forms an isotropic liquid, while at lower temperatures the material will form an anisotropic crystal. At times they show positional and directional ordering similar to crystals, while at other times (i.e with an increase in temperature) there is no positional or directional ordering and liquid crystals behave as an ordinary liquid. These two phases show very different reactions to polarized light. In the liquid crystal or anisotropic phase, the liquid crystal shows birefringence. That is polarized light has a different refractive index when it is orthogonal to the optical axis than it does when it is aligned to the optical axis. The optical axis is the average orientation along the long axis of the nematic molecules. In the liquid or isotropic phase polarized light has the same refractive index in all directions.

Nematic liquid crystals consist of thread-like molecules and will align themselves so that the long axes of neighbouring molecules point approximately in the same direction when in the liquid crystal phase. A stylized image of a nematic liquid crystal is shown in Figure 1.4. This alignment of the nematic molecules is known as the optical axis or director of the liquid crystal. Exposing nematic liquid crystals to an electric or magnetic field induces a dipole moment in the molecules. The molecules then rotate to minimise their potential energy, until the torque is balanced by the elastic forces

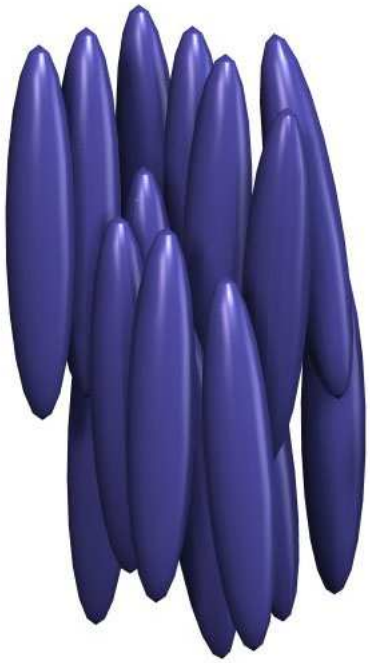

Figure 1.4: Schematic diagram of a nematic liquid crystal. The molecules are long and thin, aligning themselves along the longitudinal axis in the nematic phase. In the presence of an electric or magnetic field the molecules will reorient themselves to reduce the level of energy in the system. Image from Kebes [20]

of the medium. As the refractive index of the nematic has a nonlinear dependence on the director orientation the refractive index of the nematic changes. The electric or magnetic fields can be applied externally or be due to an optical beam.

Liquid crystals are optically highly nonlinear. Their physical properties such as temperature, molecular orientation, density and electronic structure are easily disturbed by an applied optical field [64]. Polarized light can cause an alignment or ordering when the liquid crystal is in the isotropic phase and in the anisotropic phase polarized light can also cause a realignment of the molecules. In both cases the result is a change in the refractive index.

These optical properties of nematics are due to their consisting of complex organic molecules that are based on aromatic elements (benzene rings) as their functional groups [64]. A benzene group has π electrons which are relatively free to roam. Hence, when an electric or magnetic field is applied, these electrons move, resulting in the molecule having a dipole moment. A liquid crystal molecule is shown in Figure 1.5. The benzene ring is shown as a hexagon and the π electrons are represented as the circle within the hexagon. As an optical beam is electromagnetic radiation, an optical beam will induce such a dipole in the molecules. The molecules then realign themselves to minimise their potential energy (Le Chateliers Principle). The molecules rotate until this torque is balanced by the elastic response. As stated above, this rotation results in a change in the optical dielectric constant and consequently the refractive index [42, 44, 45]. Not only the optical properties of the nematic are changed by this response. Physical properties, such as viscosity and transition temperature are also changed, which can lead to further nonlinear optical effects.

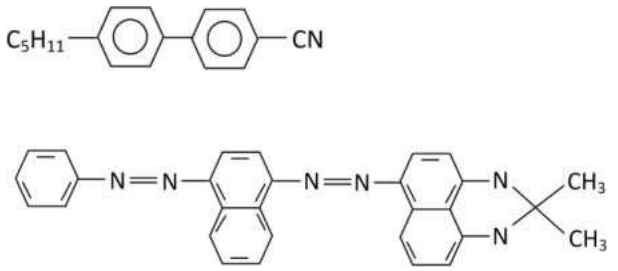

Figure 1.5: Diagrammatic representation of a nematic liquid crystal molecule (upper) and an azo dye (lower). The upper image is the molecular structure of 5CB and the lower image is Sudan Black, an azo dye that can be added to nematic liquid crystals causing them to become a defocussing medium. The nematic molecule contains two benzene rings (the hexagons with an enclosed circle). The circle inside the hexagon represents π electrons that are loosely bound. Image from [21]

In a linear regime, that is low light intensity, light does not change the orientation of the nematic molecules and so does not affect the director angle and refractive index. The beam will then diffract as in a homogeneous medium. As the power of the beam is increased, the nematic molecules will reorient themselves and the director orientation changes. This increases the refractive index with the change in the refractive index higher in areas of higher intensity of the beam. An index well is then formed by the light beam, leading to the formation of a self-focusing waveguide and a self-trapped beam or optical solitary wave, termed a nematicon. The perturbation due to the optical field extends far beyond the the beam profile owing to the elastic links between the molecules [44]

Nematicons form due to a balance between the nonlinear self-focusing and linear diffraction. The increase in the refractive index with beam intensity results in the optical beam forming its own waveguide. This process is termed self-focusing. As the optically induced torque must overcome the elastic forces for the nematic molecules to rotate, a minimum beam power is required for this rotation. This minimum power is termed the optical Freédericksz threshold [64].

If the nematic director is initially orthogonal to the electric field of the polarised light beam, the usual experimental configuration, then the optical power needed to overcome the Freédericksz threshold can be large enough to heat the nematic enough so that it undergoes a phase transition [15,17]. To lower the Freédericksz threshold, the nematic is pre-tilted by rotating the molecules so that they are not orthogonal to the electric field of the input beam [42, 44]. The Freédericksz threshold in fact vanishes when the director and electric field are at $\pi/4$ to each other [42, 44, 64]. This pre-tilt can be accomplished by either applying an external low frequency electric field in the direction of the electric field of the optical beam across the cell containing the nematic or by "rubbing". "Rubbing" consists of treating the walls of the nematic cell so that it has a static charge. The nematic molecules at the cell walls then rotate. This rotation is transmitted to the bulk via the elastic forces. With this pre-tilt, optical beams of milliwatt power can rotate the nematic molecules with no major heating of the medium. In this work, the pre-tilt will be due to an external applied electric field.

To illustrate the effect the Freédericksz threshold has on the formation of solitons in nematic liquid crystals (NLC) an optical beam with input power $4.2mW$ was passed through a nematic liquid crystal with no external static/low-frequency electric field

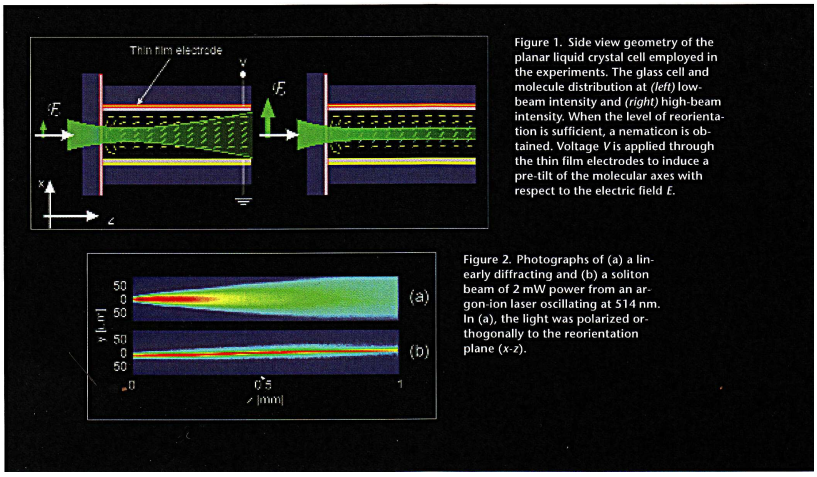

Figure 1.6: The top diagram shows the need for sufficient power in a laser beam to form a soliton in a liquid crystal. The lower photograph shows a laser beam diffraction in a liquid crystal when it is of low power. When the power is increased to $2mW$ a soliton forms. Diagram from G. Assanto, University ROMA TRE [35]

.

applied, the beam diffracted and a nematicon did not form. However, when an external field of $0.8V$ at $1kHz$ was applied across a $74\mu m$ cell, the Freédericksz threshold was lowered. An optical beam with input power $4.2mW$ was again passed through the nematic liquid crystal and now a nematicon was formed [44]. The Freédericksz threshold had been exceeded by the optical beam, so a soliton could now be created.

It must be stressed that the input beam needs be sufficiently high in power to form a soliton. This is illustrated in Figure 1.6. In the first part of the figure a laser beam is passed into a planar liquid crystal cell with an applied voltage V across it to reduce the Freédericksz threshold. The beam on the left is not intense enough to reorientate the nematic molecules and no nematicon is obtained. As the intensity of the optical beam is increased, the nematic molecules reorient and when the reorientation of the molecules is sufficient, a nematicon is formed. In the lower part of the figure there are photographs of argon-ion lasers entering a NLC showing this result experimentally. In part (a) the beam is not sufficiently powerful and the beam diffracts, while in part (b) the power of the beam is increased to $2mW$. There is now enough power to cause the nematic molecules to reorient and the beam self-focuses forming a soliton.

Figure 1.7 shows the results of an experiment where two laser beams are co-launched into a nematic liquid crystal. The upper photograph shows the solitons proceeding to the end of the cell without interaction. The laser beams have a wavelength of $514nm$ and a power of $1.7mW$. The lower photograph shows the result when the power is increased to $4.3mW$. The increased power of the solitons has caused the molecules of the liquid crystal to reorient and raise the refractive index of the cell in the vicinity of the beams. This increases the attractive force between the beams to such an extent that the beams interact and cross-over, so the upper beam has now become the lower beam and the lower beam is now the upper beam. The output positions of the beams have flipped. Through the use of a two level input (i.e. low and high power nematicons), a power controlled X-junction can be formed capable of flipping the output of two signals co-launched with solitons. Here the signal beams are using the solitons as wave-guides. If the solitons are low power then the signals proceed straight across the nematicon. If

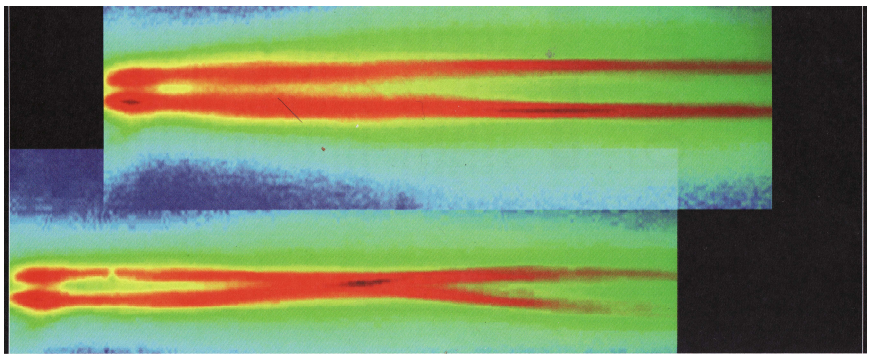

Figure 1.7: Two solitons co-propagating in a nematic liquid crystal. The top photograph is for low powered solitons $1.7mW$ with the solitons proceeding straight across the cell. The bottom photograph shows high powered solitons $4.3mW$. They cross-over and their output positions are flipped. Photograph from G. Assanto, University ROMA TRE [35]

the solitons are high power the output positions of the signal beams will be flipped.

In theory the NLS equation (1.3) can be solved exactly using the inverse scattering transform but in practice the evolution of the soliton solutions are driven by the radiation loss and this radiation loss is difficult to determine with inverse scattering. An alternate way to derive approximate solutions is through the use of a Lagrangian formulation and a trial solution. This leads to a system of ordinary differential equations (ODEs) that can incorporate the radiation loss and the solutions of these ODEs are in good agreement with numerical solutions of the NLS equation, see Kath and Smyth [91].

In a nematic liquid crystal the NLS equation must also take into account the orientation of the nematic molecules. This is done through the introduction of a Helmholtz equation that governs the angle of rotation of nematic molecules. There are known exact soliton solutions for single NLS equations, however, there is no known soliton solution for coupled focussing or defocussing NLS equations.

The simplest system of equations describing the propagation of a polarised, coherent optical beam through a nematic medium is [42, 44]

$$i\frac{\partial u}{\partial z} + \frac{1}{2}\nabla^2 u + 2\theta u = 0, \tag{1.7}$$

$$\nu\nabla^2\theta - 2q\theta = -2|u|^2. \tag{1.8}$$

Here u is the complex valued envelope of the electric field of the optical beam and θ is the optically induced rotation of the nematic molecules over the pre-tilt angle. The direction z is the propagation direction of the optical beam and x is the direction of both the electric field of the (polarised) optical beam and the external electric field. The y direction completes the coordinate triad. The Laplacian ∇^2 is in the transverse (x, y) plane. The parameter ν measures the elastic response of the nematic and is large, $O(100)$ in the experimental regime [59, 90, 91]. The parameter q is proportional to the square of the magnitude of the pre-tilt electric field. The electric field equation (1.7) is an NLS type equation and the director response equation (1.8) is a linear elliptic, Helmholtz equation. The nematic equations (1.7) and (1.8) are a $(2+1)$ dimensional system as z is mathematically a time-like direction, which can be seen on comparing

with the NLS equation (1.3). It is known from numerical and experimental results [42] that these equations have solitary wave solutions. However, there are no known exact, general analytical solutions for these solitary waves, even in $(1 + 1)$ dimensions. The only known solitary wave solutions are isolated solitary wave solutions with a fixed amplitude for fixed parameter values, both in $(1 + 1)$ and $(2 + 1)$ dimensions [74]. Again, approximate solutions can be derived through the use of a Lagrangian formulation and a trial solution. In this thesis the method used to produce solutions of coupled nematic equations is to use a Lagrangian formulation. Trial functions based on known soliton solutions of the single NLS equation are then substituted into the Lagrangian formulation and averaging is carried out over the dimensions orthogonal to the direction of propagation [45]. This leads to a system of ordinary differential equations (ODEs) called the modulation equations. It is possible to add refinements to the modulation equations that allow for effects such as the loss of radiation from the solitons or for treating the solitons as a continuous medium. The solutions of these ODEs can offer good agreement with numerical solutions of the coupled NLS equations when the most important features of the system are taken into account.

Nematic liquid crystals have proved to be an ideal medium in which to study many nonlinear optical phenomena due to their "huge" nonlinear response, which enables nonlinear effects to be observed over millimetre distances [42, 44]. Nematic liquid crystals having a focusing response to optical beams, so that the refractive index increases with the beam intensity [42], leading to self-sustaining beams for which this self-focusing balances linear diffraction [42, 45]. Nematic liquid crystals have then been found to support bulk optical solitary waves[42, 46], termed nematicons, and optical vortices[42, 44, 47]. A determining feature of nematic liquid crystals is their "nonlocal" response in that the response of the nematic medium to an optical forcing extends far beyond the waist of the optical beam[42, 44, 49]. This nonlocal response means that two or more nematic beams can interact at a distance via the nematic medium without the optical fields interacting directly as seen in Figure 1.7. This strong nonlocal interaction results in the interaction between nematicons being attractive, independent of the relative phase of the nematicons, as shown experimentally [51, 52, 53] and numerically [54], in contrast to local NLS solitons for which the interaction is attractive if the solitons are in-phase and repulsive if they are out of phase [55]. This attraction due to the nematic medium is strong enough to counteract the centrifugal repulsive force when two or more nematicons spiral around each other, so that a bound state can form [56, 57, 58, 59], in analogy with gravitational attraction [58, 59]. The mutual attraction between co-propagating nematicons, resulting in the formation of a bound state, also extends to counter-propagating nematicons [60, 61, 62]. Indeed, counter-propagating nematicons can merge to form a single beam on close enough approach [61, 62].

One of the reasons for the experimental and theoretical interest in nonlinear optical beams in nematic liquid crystals is their possible application in all-optical devices[42]. An optical beam in a nematic liquid crystal can, in principle, be routed anywhere within a liquid crystal cell. It can then act as a reconfigurable "wire" for a co-propagated signal beam. In particular, a number of mechanisms have been proposed for nematicon-based logic gates and "light" valves, based on the controlled routing of a nematicon in a liquid crystal cell. The actual routing of the nematicon can be produced via a number of control mechanisms. The simplest is through an externally applied electric field[63]. The adjustment of this external electric field can be used to control the rotation of the nematic molecules, thus changing the refractive index of the nematic [42, 64], resulting in refraction of the nematicon. Of interest to the current work, the trajectory of a nematicon can be controlled by the presence of another optical beam. This control beam

can be in a plane orthogonal to the signal beam, resulting in a "light valve"[68, 69], or can be a co-propagating nematicon in the same plane[51, 72]. In particular, the control of a signal nematicon by a co-propagating control nematicon can be used to design power dependent X, NOR and AND logic gates[72].

The present thesis details a theoretical investigation of the control of the trajectory of a nematicon, the signal beam, by another co-propagating nematicon, the control beam, as in experimental investigations of nematicon-based logic gates [51, 72]. This investigation will be based on the use of suitable trial functions for the nematicon solution in a Lagrangian formulation of the equations governing nonlinear optical beam propagation in nematic liquid crystals[73]. The use of trial functions is necessitated due to the lack of any general exact solutions for a nematicon, except isolated solutions for fixed parameter values [74], as discussed above. However, an appropriate choice of a trial function has been found to give approximate solutions in excellent agreement with full numerical solutions for a nematicon[44, 73, 74]. This holds for a range of trial functions, for example Gaussians and hyperbolic secants, as long as they are in broad agreement with numerical nematicon profiles. This approximate Lagrangian method has been previously used to study nematicon interaction [58, 76, 77], with excellent agreement found with numerical solutions.

In this study, the output position in a nematic cell of a signal beam (nematicon) will be controlled by another nematicon acting as a control beam. The input positions of both beams are fixed, as well as the input angle of the signal beam. The output position of the signal beam is controlled by varying the input angle of the control beam. The input angle of the control beam required to lead to a specific output position of the signal beam is calculated. This input angle is calculated using both full numerical solutions of the nematicon equations and approximations based on the Lagrangian method discussed above. The first Lagrangian approach treats the beams as point particles, as their detailed profiles are averaged out, and yields dynamical equations for the beam trajectories which are analogous to those for point particles moving in a potential well which is determined by the response of the nematic medium. This particle approximation has been used in previous studies of interacting nematicons and has been found to yield excellent agreement with full numerical solutions [58, 59]. In addition, these particle approximations have been found to be useful for general perturbed solitary wave problems [78]. However, in the present application to beam control this point particle approximation is found to yield only poor agreement with full numerical solutions of the nematic equations, for reasons to be discussed. The particle approximation is then extended to take account of the detailed profile of the optical beams, yielding what is termed an extended particle approximation, but which could also be termed a rigid body approximation as the beam is now treated as an extended body. This extended Lagrangian approach is found to yield results in near perfect agreement with full numerical solutions, both for the beam trajectories and the input angle of the control beam required to obtain a given output position of the signal beam. This extension of the standard Lagrangian approach should prove useful for other problems involving solitary waves and their interactions and which have, to date, been studied using the standard particle approximation only [78].

1.3 Layout of the thesis

The first part of the thesis derives approximations modelling the interaction of two optical beams in a nematic liquid crystal. The models are then used to calculate tra-

jectories of the optical beams in the liquid crystal. One of the beams (the signal beam) is guided to a target area at the end of the liquid crystal cell by varying the input angle of the other beam (the control beam). Both of the models are derived from Lagrangian formulations with suitable trial functions substituted into the Lagrangian. The accuracy of the models are then determined by comparison with solutions calculated from the numerical solutions of the coupled NLS equations for optical beams in a nematic liquid crystal. Chapter 2 examines a model called the two body particle approximation, while in Chapter 3 another model, called the extended particle model, is examined.

Chapter 2 examines how two co-propagating nematicons in a liquid crystal interact with each other. We want one nematicon to be sent to a target area by varying the input angle of the other nematicon. The governing equations for two solitons in a nematic liquid crystal are presented and the numerical methods to solve these equations are discussed. The results from these numerical methods are the basis for determining the accuracy of any approximate model. The approximate model is determined using a Lagrangian formulation. Substituting trial functions into the Lagrangian leads to the derivation of the modulation equations for the approximate model. The numerical methods used to produce solutions from the modulation equations are examined. The numerical methods are used iteratively until the desired initial angle is found. To speed the process of finding the initial angle a routine is developed that estimates the initial angle given the data from previous trajectories. The results from the two body particle approximation are presented and compared to the results from numerical solutions of the coupled NLS equations. The approximate model showed a weakness in the manner that the attractive force between the two nematicons is calculated. The attractive force was smaller than that calculated by numerical solutions of the coupled NLS equations.

Using the modulation equations derived from the two body particle approximation, the equation for position can be thought of as the equivalent, in optics, of the conservation of momentum for a particle in classical mechanics. The intensity of the optical beam is then the "mass" of the particle. This analogy forms the foundation for the equations describing the next approximate model.

Another analogy is made with classical mechanics, the force between two bodies as given by Newton's law of universal attraction. This force is given by $G\frac{m_1 M_2}{r^2}$, where G is the universal gravitational constant, m_1 is the mass of body 1, M_2 is the mass of body 2 and r is the distance between the bodies. It can be shown that with an inverse square law, when two solid bodies have a uniform density, the force of attraction between the two bodies can be considered to be as if all the mass is concentrated at the centre of mass of each body [118]. In the case of two nematicons, the attraction is not an inverse square law, but Gaussian in nature and the density is related to the profile of the beam. Taking these into consideration, the attraction between two solid nematicons cannot be assumed to be concentrated at the centre of mass, but needs to be calculated according to the attraction between the nematicons and the density of the nematicons.

Chapter 3 presents the derivation of the next approximate model. It starts with the optical equivalent of the conservation of momentum equation in classical mechanics and adjusts the attractive force between the nematicons according to the previous discussion. That is the model treats the solitons as a continuum rather than a particle and the model is called the extended particle model. Results are shown for the trajectories calculated from the two body particle model, the extended particle model and the numerical analysis of the coupled NLS equations. Also shown are comparisons of the initial angle solutions given by the two body particle model, the extended particle model and numerical solutions of the coupled NLS equations. The results from the extended particle model are in excellent agreement with numerical solutions of the

coupled NLS equations.

Chapter 4 examines the interaction between two dark nematicons in a defocusing nematic cell. A nematic is usually a focusing medium, but can be made defocusing through the addition of azo dyes [70]. It is found that when the two beams are of different wavelength, so that their diffraction coefficients are different, an instability develops, as for a local medium [74]. However, when the beams have the same wavelength, so that the diffraction coefficients are the same, the beams are stable and oscillate in position and amplitude about each other.

The approximate model is again based on a Lagrangian formulation. After substituting trial functions into the Lagrangian the modulation equations are derived [45]. The numerical techniques used to solve the modulation equations are detailed. Some results are predicted by analysing the modulation equations. The first is that beams with the same diffraction coefficient (same wavelength) will oscillate in position and amplitude around each other. This is confirmed by the results from the approximate model and from numerical solutions of the coupled nematic equations. The second prediction is that for dark nematicons with unequal diffraction coefficients an instability results. Again, this is confirmed from numerical solutions of the coupled nematic equations. However, the modulation equations do not predict this instability. This is because it is totally radiation driven as the nematicons evolve with their shed radiation. Lagrangian methods tend to be poor when there are such regimes.

Finally Chapter 5 gives a general analysis of the results, main outcomes, methodology, conclusions and possible future research arising out of the results from this thesis.

Chapter 2

Two interacting optical beams

2.1 Controlling laser beams

The aim of this thesis is to investigate the interaction of optical beams in nematic liquid crystals. The first scenario to be studied is a laser beam being controlled by another co-propagating laser beam. The idea is for the first beam, called the signal beam, to be guided to a target area at the far end of a liquid crystal cell by varying the input angle of the second beam, called the control beam. The two laser beams need to be of sufficient power to be able to form nematicons. The presence of the nematic liquid crystal means the influence of an optical beam extends far beyond its width and is called the nonlocal effect [41]. Nematic liquid crystals will rotate in the presence of an electric field and this rotation will extend the influence of the laser beams beyond their widths [44]. This is called the nonlocal effect of the nematic liquid crystal and is measured by a parameter ν. The larger the value of ν, the larger is the nonlocal effect. Typical values of of ν in the experimental regime are $O(100)$[49, 80, 81]. This strong nonlocal interaction results in the interaction between nematicons being attractive, independent of the relative phase of the nematicons, as shown experimentally[51, 52, 53] and numerically[54], in contrast to local NLS solitons for which the interaction is attractive if the solitons are in-phase and repulsive if they are out of phase[55]. This attraction due to the nematic medium is strong enough to counteract the centrifugal repulsive force when two or more nematicons spiral around each other, so that a bound state can form[56, 57, 58, 59], in analogy with gravitational attraction[58, 59]. The mutual attraction between co-propagating nematicons, resulting in the formation of a bound state, also extends to counter-propagating nematicons[60, 61, 62]. Indeed, counter-propagating nematicons can merge to form a single beam on close enough approach[61, 62].

In detail, there are two coherent light beams of the same wavelength entering a cell filled with nematic liquid crystal. The electric fields of the laser beams are polarised in the same direction, which is taken to be the x direction. The two beams are co-propagating, with the direction of propagation taken as the z direction and the intensity of the laser beams is high enough to form solitary waves, termed nematicons [44]. The cell is of length L in the z direction. An external low frequency electric field is applied across the cell in the x direction. This field is known as a pre-tilt and has the effect of aligning the nematic liquid crystal molecules to an initial angle of θ_0 to the z direction and also of lowering the Fréedericksz transition threshold. In the presence of optical beams, the electric fields of these beams cause a further rotation of the nematic liquid crystal molecules of an angle θ, so that the molecules in the liquid crystal have a total

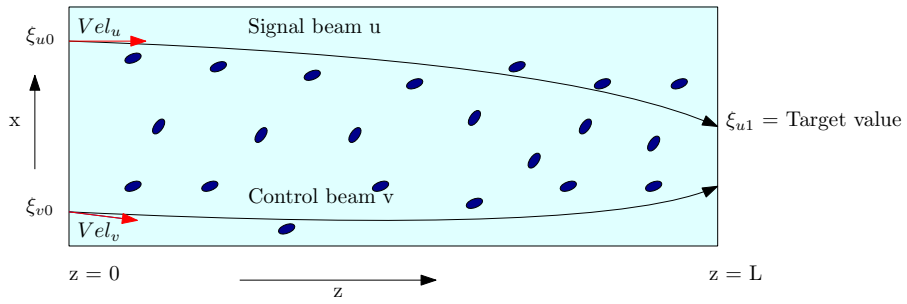

Figure 2.1: Laser beams in liquid crystals.

angle of rotation of $\theta_0 + \theta$ to the z direction. Due to the pre-tilt, solitons can then form at low optical powers, of the order of milliwatts, so that $|\theta| \ll \theta_0$ [42, 44].

By definition the upper laser beam is labelled the signal beam and the lower beam the control beam, see Figure 2.1. Both the signal and control beams enter the liquid crystal cell at fixed positions designated as ξ_{u0} and ξ_{v0}, respectively. The input angle of the signal beam is fixed at 0 degrees and the input angle of the control beam is varied until the signal beam hits the target area at the end of the cell. The angles of the beams are mathematically velocities in the analysis, where the velocity is defined as $\frac{dx}{dz} = \tan(\phi)$ and the input angle of the beam is ϕ. This is because z is a time-like variable. Using these definitions Vel_u is always 0 and Vel_v is $\tan(\phi_v)$.

The approach taken to study the interaction of nematicons in a liquid crystal is to first look at the equations that describe the behaviour of optical beams in nematic liquid crystals. The equations are then solved numerically using techniques that have been proven to give accurate results. These results can then be used as a standard so that the effect of approximations used in other models can be evaluated.

2.2 Equations describing optical beams in a nematic liquid crystal

The equations governing the electric field of a single optical beam and the angle of rotation of the long axis of a nematic molecule, termed the optical axis, or director, in the presence of an applied static electric field are a set of equations consisting of a nonlinear Schrödinger (NLS) equation for the beam and a Poisson equation for the director [35, 53, 63]. A full derivation of these equations from Maxwell's equations can be found in [44, 71]. The nematicon equations governing the co-propagation of two polarised beams of coherent light of the same wavelength in a cell filled with nematic liquid crystals were derived in Alberucci *et al.* [34] and are

$$i\frac{\partial u}{\partial z} + \frac{1}{2}\nabla^2 u + u\sin(2\theta) = 0, \tag{2.1}$$

$$i\frac{\partial v}{\partial z} + \frac{1}{2}\nabla^2 v + v\sin(2\theta) = 0, \tag{2.2}$$

$$\nu\nabla^2\theta - q\sin 2\theta = -2(|u|^2 + |v|^2)\cos(2\theta). \tag{2.3}$$

Here the Laplacian ∇^2 is in the (x, y) plane. The variables u and v are the complex valued envelopes of the electric fields of the laser beams. The variable q is related to the electric field applied across the cell as a pre-tilt and for a laser beam of amplitude A is defined as

$$q = \frac{4\Delta\epsilon_{RF}}{\epsilon_0 n_a^2 A^2 \tan 2\psi_0} E^2, \tag{2.4}$$

where $\Delta\epsilon_{RF}$ is the low-frequency anisotropy, $n_a^2 = n_\parallel^2 - n_\perp^2$ is the optical anisotropy ($n_\parallel$ and $n_\perp$ being the refractive indices for an optical beam parallel and normal to the director alignment [63], and E is the strength of the applied static electric field [53]. The variable ν measures the elastic response of the nematic and hence measures the nonlocality of the liquid crystal. Large nonlocality corresponds to a large value of ν, $\nu = O(100)$ in experiments [59, 90, 91] . The angle θ is the additional rotation from the pre-tilt angle θ_0 caused by the applied electric field and $|\theta| \ll |\theta_0|$ for milliwatt beams.

Equations (2.1) and (2.2) are coupled nonlinear Schrödinger equations for each of the nematicons and equation (2.3) is a director (Poisson) equation that governs the reorientation of the optical axis. For nonlocal regimes ν is large and assuming small amplitudes of the beams ie $|\theta| \ll |\theta_0|$, the governing equations can be approximated by

$$i\frac{\partial u}{\partial z} + \frac{1}{2}\nabla^2 u + 2u\theta = 0, \tag{2.5}$$

$$i\frac{\partial v}{\partial z} + \frac{1}{2}\nabla^2 v + 2v\theta = 0, \tag{2.6}$$

$$\nu\nabla^2\theta = 2q\theta - 2(|u|^2 + |v|^2). \tag{2.7}$$

For simplicity, the cell will be taken to be much wider than the beams. The effect of the boundaries can then be neglected and it can be assumed that $u \to 0$, $v \to 0$ and $\theta \to 0$ as $x^2 + y^2 \to \infty$. Typical beam widths are $3\mu m$ and typical cell widths are $75\mu m$ [46], so these are reasonable assumptions. Unlike the NLS equation (1.3), there is no exact solution of the system of equations (2.5) – (2.7), so alternate methods of solution are needed to study the interaction between the laser beams.

The system of equations (2.5)–(2.7) has been stated for optical beams in a nematic liquid crystal, but is in fact general. It governs nonlinear optical beam propagation in media for which the nonlinearity is coupled to some diffusive phenomena [82], examples being thermal media [83], lead glasses [84, 85, 86] and certain photo-refractive crystals [87]. In addition, a similar system of equations arises in simplified α models of fluid turbulence [88, 89].

Equations (2.5)–(2.7) are the basic equations describing coupled nematicon evolution and will be used to investigate a wide range of nematicon behaviour. There is no general solution of this set of equations. To study the behaviour of nematicons one can solve the equations numerically, which provides few insights into the dynamics of nematicon evolution and its underlying mechanism, yet provides an accurate portrayal of beam evolution, or one can solve the governing equations approximately. Both of these will be done in this thesis.

2.3 Pseudo-spectral numerical scheme for the two soliton system

Since it is not possible to obtain general exact solutions of the system of nematic equations (2.5) – (2.7) it is important to have a standard to which the results from any models that have used approximations in their development can be compared. The standard must be demonstratively accurate compared with other possible methods of producing results from the same set of equations.

The method based on that described in Fornberg & Whitham [115] is used to numerically integrate the nematic equations. The results from this method have been compared with the results obtained using the same equations with a method based on the Dufort-Frankel finite difference scheme to solve the NLS equation (2.5) and a Gauss-Siedel iteration with successive over relaxation to solve the director equation (2.7). Identical step sizes were taken for each method and the agreement was found to be excellent. These results were described in a paper by García-Reimbert et alia [38].

The original method described by Fornberg & Whitham [115] was intended to solve a single NLS equation in the local limit. The method needs to be extended to solve the equations for two nematicons in the nonlocal limit, the equations being

$$i\frac{\partial u}{\partial z} + \frac{1}{2}D_u\nabla^2 u + 2\theta u = 0, \tag{2.8}$$

$$i\frac{\partial v}{\partial z} + \frac{1}{2}D_v\nabla^2 v + 2\theta v = 0, \tag{2.9}$$

$$\nu\nabla^2\theta - 2q\theta = -2(|u|^2 + |v|^2), \tag{2.10}$$

where ∇^2 is the Laplacian over the (x, y) plane and z is a time-like variable. Here the constants D_u and D_v are the diffusivity for the beams u and v.

Taking the double Fourier transform of the NLS equations, (2.8) and (2.9), in the variables x and y produces the equations

$$\frac{d\bar{u}}{dz} + \frac{i}{2}D_u(w_x^2 + w_y^2)\bar{u} - iF\{2\theta u\} = 0, \tag{2.11}$$

$$\frac{d\bar{v}}{dz} + \frac{i}{2}D_v(w_x^2 + w_y^2)\bar{v} - iF\{2\theta v\} = 0. \tag{2.12}$$

Here $\bar{u}$ is the Fourier transform of u and $F\{2\theta u\}$ is the Fourier transform of $2\theta u$, with similar definitions for the variable v, while w_x and w_y are the Fourier transform variables for x and y. The double Fourier transform of $u(x, y, z)$ over the variables x and y is defined as

$$\bar{u}(w_x, w_y, z) = \tfrac{1}{2\pi}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} u(x, y, z)e^{iw_x x}e^{iw_y y}dxdy, \tag{2.13}$$

Using an integrating factor equation (2.11) can be rewritten as

$$\frac{d}{dz}\left(\bar{u}e^{iD_u(w_x^2+w_y^2)z/2}\right) = (iF\{2\theta u\})e^{iD_u(w_x^2+w_y^2)z/2}. \tag{2.14}$$

This equation can be solved using Fast Fourier Transforms (FFT), [114], by setting up a grid over the (x, y) plane. The range of the grid in the x direction is L_x, defined as

the difference between the maximum value of x and the minimum value of x. L_x is divided into N_x grid points, where N_x is of the form 2^k, k an integer. A similar grid is formed for the y dimension, with the range being L_y and the number of grid points N_y. Equation (2.14) is discretised as the equation

$$\frac{d}{dz}\left(\bar{u}_{jk}e^{iD_u(w_x^2+w_y^2)z/2}\right) = (iF\{2\theta_{jk}u_{jk}\})e^{iD_u(w_x^2+w_y^2)z/2}. \tag{2.15}$$

Here $\bar{u}_{jk}$ is the FFT of u, θ_{jk} is the value of θ and u_{jk} the value of u at the grid point $x = j, y = k$. w_x, w_y are given by,

$$w_x = \frac{2\pi j}{L_x}, \quad j = \frac{-N_x}{2}+1,\ldots,\frac{N_x}{2}, \tag{2.16}$$

$$w_y = \frac{2\pi k}{L_y}, \quad k = \frac{-N_y}{2}+1,\ldots,\frac{N_y}{2}. \tag{2.17}$$

Equation (2.15) is an ordinary differential equation in Fourier space for which the only unknown variable is θ_{jk}. The director equation (2.10) is solved for θ and can be written as

$$\nabla^2\theta - \frac{2q}{\nu}\theta = \rho(x,y,z), \tag{2.18}$$

where

$$\rho(x,y,z) = -\frac{2}{\nu}(|u|^2+|v|^2). \tag{2.19}$$

Using finite differences to approximate ∇^2, equation (2.18) is rewritten as

$$\frac{\theta_{j+1,k}-2\theta_{j,k}+\theta_{j-1,k}}{\Delta_x^2} + \frac{\theta_{j,k+1}-2\theta_{j,k}+\theta_{j,k-1}}{\Delta_y^2} - \frac{2q}{\nu}\theta_{j,k} = \rho_{j,k}. \tag{2.20}$$

Here Δ_x is defined as L_x/N_x and Δ_y is defined similarly.

The discrete Fourier transform in x and y of a variable θ, [114], is given by

$$\bar{\theta}_{m,n} = \sum_{j=0}^{N_x-1}\sum_{k=0}^{N_y-1}\theta_{j,k}e^{2\pi imj/N_x}e^{2\pi ink/N_y}, \tag{2.21}$$

and the discrete inverse Fourier transform by

$$\theta_{j,k} = \frac{1}{N_xN_y}\sum_{m=0}^{N_x-1}\sum_{n=0}^{N_y-1}\bar{\theta}_{j,k}e^{-2\pi imj/N_x}e^{-2\pi ink/N_y}. \tag{2.22}$$

Substituting this Fourier transform (2.21) into equation (2.20) and taking $\Delta_x = \Delta_y = \Delta$ produces the result, [114]

$$\bar{\theta}_{m,n} = \frac{\Delta^2\bar{\rho}_{m,n}}{2\left(\cos\frac{2\pi m}{N_x}+\cos\frac{2\pi n}{N_y}-2\right)-\frac{2q\Delta^2}{\nu}}. \tag{2.23}$$

Equations (2.20) and (2.22), along with the numerical efficiency inherent in the fast Fourier transform process, provides a mechanism to convert an array of variables

into their Fourier transforms and to also be able to reverse this process, so an array of Fourier transforms can be quickly converted into the original variables. The efficiency of this process in both directions will be used to solve the equations for two nematicons [114].

$$\theta_{j,k} \underset{\text{inverse FFT}}{\overset{\text{FFT}}{\rightleftharpoons}} \bar{\theta}_{m,n}. \tag{2.24}$$

Equation (2.15) and the corresponding equation for v can be written

$$\frac{d}{dz}\left(\bar{u}_{jk}e^{iD_u(w_x^2+w_y^2)z/2}\right) = G_1(z,u,\theta), \tag{2.25}$$

$$\frac{d}{dz}\left(\bar{v}_{jk}e^{iD_v(w_x^2+w_y^2)z/2}\right) = G_2(z,v,\theta). \tag{2.26}$$

where

$$G_1(z,u,\theta) = 2iF\{\theta_{jk}u_{jk}\}e^{iD_u(w_x^2+w_y^2)z/2}, \tag{2.27}$$

and

$$G_2(z,v,\theta) = 2iF\{\theta_{jk}v_{jk}\}e^{iD_v(w_x^2+w_y^2)z/2}. \tag{2.28}$$

Here $F\{\theta_{ij}u_{ij}\}$ is the Fourier transform of $\theta_{ij}u_{ij}$.

Equations (2.25) and (2.26) are a system of first order ordinary differential equations that will be solved using the standard fourth order Runge-Kutta routine.

Defining the time step Δz as L/N_z where L is the length of the nematic crystal cell and N_z is the number of time steps to be taken, then the time like variable z_j is defined as $z_0 + hj$, where j is an integer taking the values $0 \ldots N_z$. Letting u_j and v_j be the value of u and v at time z_j, then the standard fourth order Runge-Kutta routine for the system (2.25) and (2.26) is carried out by calculating the intermediate steps,

$$k_{1,1} = \Delta z G_1(z_j, u_j, \theta_{j,1}), \tag{2.29}$$

$$k_{1,2} = \Delta z G_2(z_j, v_j, \theta_{j,1}), \tag{2.30}$$

$$k_{2,1} = \Delta z G_1(z_j + \frac{1}{2}\Delta z, u_j + \frac{1}{2}k_{1,1}, \theta_{j,2}), \tag{2.31}$$

$$k_{2,2} = \Delta z G_2(z_j + \frac{1}{2}\Delta z, v_j + \frac{1}{2}k_{1,2}, \theta_{j,2}), \tag{2.32}$$

$$k_{3,1} = \Delta z G_1(z_j + \frac{1}{2}\Delta z, u_j + \frac{1}{2}k_{2,1}, \theta_{j,3}), \tag{2.33}$$

$$k_{3,2} = \Delta z G_2(z_j + \frac{1}{2}\Delta z, v_j + \frac{1}{2}k_{2,2}, \theta_{j,3}), \tag{2.34}$$

$$k_{4,1} = \Delta z G_1(z_j + \Delta z, u_j + k_{3,1}, \theta_{j,4}), \tag{2.35}$$

$$k_{4,2} = \Delta z G_2(z_j + \Delta z, v_j + k_{3,2}, \theta_{j,4}), \tag{2.36}$$

enabling the value of u at time step z_{j+1} to be calculated as

$$u_{j+1} = u_j + \frac{1}{6}\left(k_{1,1} + \frac{1}{2}k_{2,1} + \frac{1}{2}k_{3,1} + k_{4,1}\right). \tag{2.37}$$

A similar formula applies for the value of v_{j+1}. The value of θ used in each intermediate step is the solution of the equation (2.23) with the function $\rho(x, y, z)$ replaced by

$$\rho(x, y, z) = -\frac{2}{\nu}(|u + \delta u|^2 + |v + \delta v|^2). \tag{2.38}$$

The various values of δu and δv used for θ in the intermediate values are shown in Table 2.1. The values of $|u|^2$ and $|v|^2$ are calculated for each point (j, k) on the (x, y) grid. These are from the initial profile of each beam or are a result of the previous iteration.

θ	δu	δv
$\theta_{j,1}$	0	0
$\theta_{j,2}$	$\frac{1}{2}k_{1,1}$	$\frac{1}{2}k_{1,2}$
$\theta_{j,3}$	$\frac{1}{2}k_{2,1}$	$\frac{1}{2}k_{2,2}$
$\theta_{j,4}$	$k_{3,1}$	$k_{3,2}$

Table 2.1: The values of δu and δv used to calculate θ

Equation (2.15) can now be used to obtain the value of $\bar{u}$ at the next time step

$$\bar{u}_{z+1} = \left[\bar{u}_z + \frac{1}{6}(\bar{k}_{1,1} + 2\bar{k}_{2,1} + k_{3,1} + \bar{k}_{4,1})\right] e^{-iD_u(w_x^2+w_y^2)\Delta z/2}. \tag{2.39}$$

The value of u_{z+1} can be found as the discrete inverse Fourier transform using routine (2.24) with v_{z+1} being found in the same way using the equation for the v beam.

2.4 Approximate models

The evolution of two coupled nematicons are governed by equations (2.5) to (2.7). As mentioned previously these equations have no known solution, so a solution is calculated using numerical methods. A natural question to ask is why should an approximate model be developed if a solution has already been calculated. The computing time required to produce the full numerical solutions is a matter of hours depending on factors such as the speed of the computer, step sizes used in the numerical method and the number of iterations needed to produce the solution. The computing time will almost certainly be reduced using an approximate model (the approximate models used in this project required at most a few minutes to produce a solution) however this is not the main reason for using an approximate model. The reason is the approximate model gives key insights into important features of the problem being studied.

Approximate models are not developed in isolation to analytic results and the results calculated from full numerical methods. The solution to equation (1.3) given by equation (1.4) provides information on the form of the solution to equations (2.5) to (2.7). The solution can then be used in the approximate model with suitable parame-

ters to give some flexibility. Conservation laws are also are also important input to the approximate models. These provide important physical restrictions to the solutions. The approximate models used in this thesis are based on a Lagrangian formulation or conservation equations derived from the Lagrangian formulation, so there is some assurance that the solution calculated from the approximate model will conform to physical restraints inherent in the problem being studied.

Once an approximate model has been developed the results are be compared to the results obtained using full numerical methods on the governing equations. If there is wide divergence between the solutions then this is an indication that an important feature of the problem has not been incorporated into the approximate model. An approximate model is developed in this chapter called the "two bodied particle approximation". When the results from this model were compared to the results obtained from the full numerical calculation of equations (2.5) to (2.7) the comparison was poor. This indicated that an important feature was not included in the approximate model. The laser beams were not being treated as solitons. Using this information the approximate model was changed so to include this feature. The results from this model are shown in Chapter 3 and the comparison between the results from new approximation and the results obtained from the full numerical calculations of equations (2.5) to (2.7) is greatly improved.

The form of the solution used in an approximate model is vital. If the solution is not correct then it is unlikely the approximate model will produce acceptable output. Input from analytic solutions and from numeric results are used to suggest the likely form of the solution. This thesis is concerned with solitons so the solution must be a soliton. Parameters of the solution are incorporated into the solution to give some flexibility. The models used in this thesis have parameters such as amplitude, width and phase that depend on the time like z dimension. Substituting a trial solution into a Lagrangian formulation and then taking variations results in a system of differential equations, known as the modulation equations. Solving the modulation equations then shows how these parameters evolve. An example of this development methodology is shown in Chapter 4.

Validation of the output form the approximate models uses limiting conditions such as the transition from non-local media to local media ($\nu \to 0$). Sometimes the solution in the limiting case is known analytically and provides a good check to the solution of the approximate model.

In summary approximate models are not developed to reduce the computer time used to calculate solutions but to increase the understanding of the solutions to the problem being studied and how the parameters in the solutions interact with each other.

2.5 Calculating the trajectories of the beams

The idea behind controlling optical beams in a nematic liquid crystal is to guide an optical beam to a target area by altering the input angle of a second beam which interacts with the first beam. Given system parameters such as the starting positions of the beams in the nematic liquid crystal cell, the input angles and the length of the cell, it is possible to guide an optical beam to only target areas lying within a finite range. It is not possible to guide the beam to any area at the end of the liquid crystal cell as there is a limit to how much the control beam can bend the signal beam. The beams may cross multiple times before they reach the end of the cell, so the trajectory of a beam may not be unique even when a given target area is reached. To prevent

this situation from occurring only trajectories that do not cross are considered in this thesis. The force between two coherent co-propagating beams is attractive, so it is only possible for the signal beam to be guided to a target area that is less than the value the beam would reach at the end of the liquid crystal cell if uncontrolled (see Figure 2.1).

A typical scenario, as illustrated in Figure 2.1, is the optical beams are started at positions ξ_{u0} and ξ_{v0} in the liquid crystal cell with the z coordinate 0. The initial angle of the u beam is $V_{u0} = 0$ and the initial angle of the v beam is V_{v0}. In order for the beams to interact, they need to form nematicons, so the amplitudes of the beams a_u and a_v need to be sufficiently high so that the beams do not decay before reaching the end of the cell ($z = L$). The value of ν will affect how much interaction there is between the two beams and typically is set to 250. The larger the value of ν, the more nonlocal the nematic is and the stronger the interaction. The variables used to describe the system environment need to be consistent between the different models, so that any comparison between the models is valid. For example the starting positions, amplitudes and angles of the input beams, the value of ν and the length of the liquid crystal cell need to be the same. The trajectories of the two beams are calculated and the value of the signal beam at the end of the liquid crystal cell ($\xi_u(z = L) = A_i$) is compared with the target value (T). If A_i does not equal T, then the input angle of the control beam, V_{v0}, is changed and the trajectories of the two beams are recalculated. This process is repeated until A_i is sufficiently close to the target value T, or the process has been repeated 20 times.

It was noted above there is a need to be able to predict the input angle that sends the signal beam to the target area. The procedure to determine the new angle uses a weighted least squares to fit a quadratic equation to the curve relating the input angle and the target value. The weights w_i are the absolute values of the inverse of the distance between the target value and the actual value the u beam reached at the end of the liquid crystal cell. In mathematical form

$$w_i = \frac{1}{|T - A_i|}, \tag{2.40}$$

where w_i is the weight for iteration i, T is the target value and A_i is the position at the end of the cell that the u beam has reached after the ith iteration.

Given the position at the end of the cell, A_i, and the corresponding input angle of the v beam, $V_v = x_i$, the least squares curve between A_i and x_i is found by the following procedure. If we want to fit the curve

$$A = a + bx_i + cx_i^2, \tag{2.41}$$

then the values of a, b and c are calculated to minimise the square of the distance between A, the predicted value, and A_i the actual value.

The square of the vertical distance, between A and A_i, is given by

$$L_S = \sum_i \left(A_i - (a + bx_i + cx_i^2)\right)^2 w_i. \tag{2.42}$$

The values of a, b and c are calculated by taking the partial derivatives of L_S with respect to a, b and c and setting these partial derivatives to 0. The resulting equations are then solved, to give the values of a, b and c that minimise the total squares of the

Iteration	End of cell	Input angle	Absolute error
1	24.56308170	0.10000000	16.56308170
2	2.85775315	-0.06563082	5.14224685
3	8.23209683	-0.02639093	0.23209683
4	8.00179578	-0.02810963	0.00179578
5	8.00001197	-0.02812302	0.00001197
6	8.00000000	-0.02812311	0.00000000

Table 2.2: Sequence of values when calculating the trajectory of an optical beam

distance.

A typical set of data is shown in Table 2.2. The input parameters are $\xi_u = 20.0$ and $\xi_v = -20.0$, the length of the liquid crystal cell is $L = 100.0$, $\nu = 200$, the amplitude of both beams is 3.0 and the widths of both beams is 2.0. The target value is 8.0. In this example it took 6 iterations to find the input angle that guided the signal beam to the target value (in this case 8.0).

For the curve in equation (2.42), taking the partial derivatives gives

$$\begin{aligned}
\frac{\partial(L_S)}{\partial a} &= -2\sum_i \left[A_i - (a + bx_i + cx_i^2)\right] w_i, \\
\frac{\partial(L_S)}{\partial b} &= -2\sum_i \left[A_i - (a + bx_i + cx_i^2)\right] x_i w_i, \\
\frac{\partial(L_S)}{\partial c} &= -2\sum_i \left[A_i - (a + bx_i + cx_i^2)\right] x_i^2 w_i.
\end{aligned}$$

Setting these to zero gives the set of equations

$$\begin{aligned}
\sum_i A_i w_i - a\sum_i w_i - b\sum_i x_i w_i - c\sum_i x_i^2 w_i &= 0, \\
\sum_i x_i A_i w_i - a\sum_i x w_i - b\sum_i x_i^2 w_i - c\sum_i x_i^3 w_i &= 0, \\
\sum_i x_i^2 A_i w_i - a\sum_i x^2 w_i - b\sum_i x_i^3 w_i - c\sum_i x_i^4 w_i &= 0.
\end{aligned}$$

Then solve these equations for a, b and c using Gaussian elimination. At least three iterations are needed before this routine will produce values for a, b and c.

After the first iteration the input angle x_2 is set as,

$$x_2 = x_1 - 0.03(T - A_1). \tag{2.43}$$

For the second iteration the input angle, x_3, is set as,

$$x_3 = \frac{T - b}{a}, \tag{2.44}$$

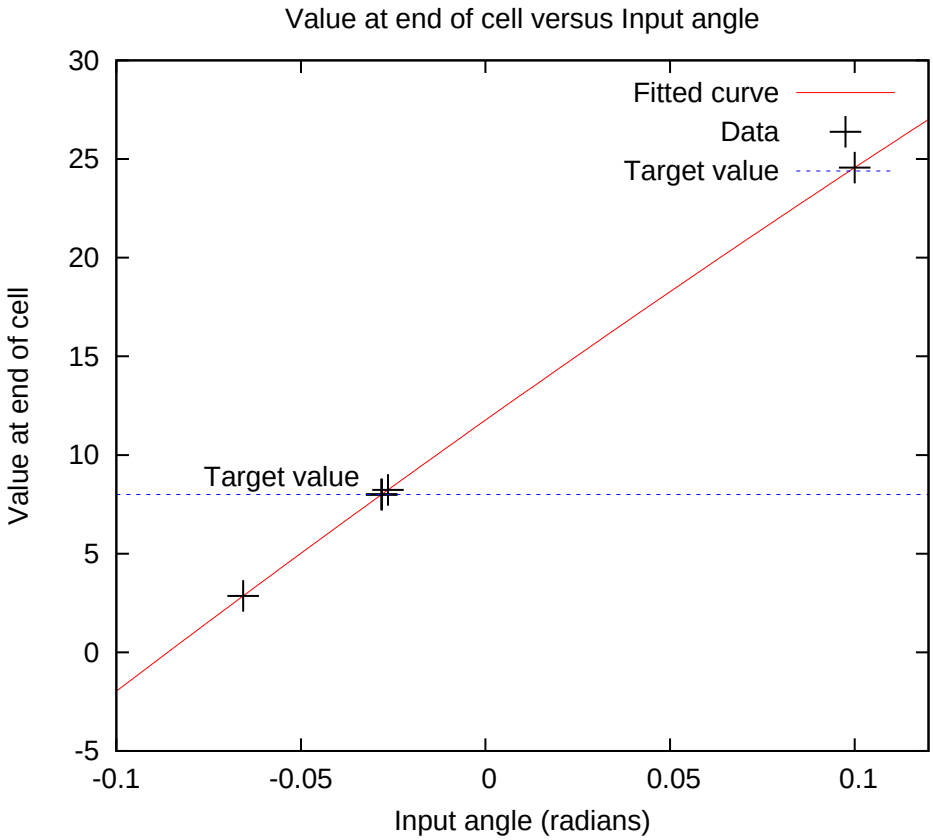

Figure 2.2: Least squares curve of input angle and value at the end of the cell.

where

$$a = \frac{A_2 - A_1}{x_2 - x_1} \quad \text{and } b = \frac{A_1 x_2 - A_2 x_1}{x_2 - x_1}. \tag{2.45}$$

Here, A_i is the actual position of beam u at the end of the cell after iteration i and x_i is the input angle of beam v at the start of iteration i.

Setting the least squares curve to the target value and solving for x predicts the next velocity.

$$a + bx + cx^2 = T, \tag{2.46}$$

so that

$$x = \frac{-b \pm \sqrt{b^2 - 4(a - T)c}}{2c}. \tag{2.47}$$

The data shown in Table 2.2 produced the curve shown in Figure 2.2, for the parameters $a = 11.7677$, $b = 132.665$ and $c = -47.1107$.

There are many possibilities for the method to calculate the initial angle of the control beam to send to signal beam to the target value. Probably the simplest is the tangent method. This consists of calculating the gradient between the actual values and

the initial angles. The method only uses the data from the last two iterations calculated with data from the other iterations not being used. No account of the distance between the actual values and the target values is taken into account , so predictions for the initial angle can diverge if the predicted initial angles are not close to the angle that will send the beam to the target value. This can be especially true if the initial angles do not straddle the target value. Another possibility is to use least squares estimation for the initial angles. This has the advantage of using all the data from the iterations. Again no allowance is made for how close the data is to the target value. The data can be weighted by using the reciprocal of the distance from the target value. The data closest to the target value has more information value when predicting the initial angle. In the end, weighted least squares were used because this is the standard way to solve an over determined problem and weights the data according to how close the laser beam is to the target value. The closer the actual value is to the target value, the more valuable that information is in predicting the next input angle. Rather than use a linear relationship between the initial angle and the actual value it was decided to use a quadratic relationship. This requires a little extra work to solve the equations and calculate the parameters, with the advantage of more flexibility. In practice the range of the input angle is small so that the quadratic equation looks linear. See Figure 2.2. No studies were undertaken to see how quickly these methods converged to the initial angle that caused the laser beam to hit the target value, however the average number of iterations was approximately six. This routine is used in all the models of this thesis that calculate the trajectories of the beams, the numerical solutions, the particle approximation model and the extended particle approximation model.

2.6 Two body particle approximation

The two body particle model is based on a Lagrangian formulation of the equations governing two nematicons in a nematic liquid crystal (2.8)–(2.10). Trial functions that approximate the solutions of the equations are substituted into the formulation. The Lagrangian is averaged over the x and y variables and variations are taken for the parameters that make up the trial functions. This is called the method of averaged Lagrangians [45] and produces a system of ordinary differential equations (ODEs) that are called modulation equations. This technique has the potential to be inaccurate if the trial function is not of the right form, or if key features of the beam evolution or form are not incorporated into the trial functions. The variational method has been used as a tool to study dynamical systems for centuries, being originally devised to study the solar system [43]. In the field of optical solitons governed by the NLS equation it was first applied by Anderson in 1983 [39, 40] and the method has been used in different areas of optics ever since [39].

In the variational method the governing equations are rewritten in their equivalent Lagrangian formulation, $L(u_z, u_{\mathbf{x}}, u)$, where $\mathbf{x}$ represents the spacial coordinates and z is the time-like coordinate. The variational method is based on the principle of stationary action where the action of a system is defined as

$$A = \int_{z_0}^{z_1} L dz. \tag{2.48}$$

The principle of stationary action states the actual path followed is the path that produces the lowest value of the action. This principle, along with the calculus of

variations, allows the actual paths followed by the system to be calculated through a system of differential equations. Lagrangian formulations of the nematic equations have been used to find approximate evolution equations for an evolving nematicon or nematicons[73, 76, 77], these equations being termed modulation equations[45].

The two colour nematic equations (2.5)–(2.7) have the Lagrangian

$$L = i\left(u^*u_z - uu_z^*\right) - D_u|\nabla u|^2 + 4\theta|u|^2 + i\left(v^*v_z - vv_z^*\right) - D_v|\nabla v|^2 + 4\theta|v|^2 - \nu|\nabla\theta|^2 - 2q\theta^2. \tag{2.49}$$

The term "two colour" in this context means that the beams have different wavelengths and possibly different diffusion coefficients D_u and D_v.

Standard modulation theory[45] is based on the periodic wave or solitary wave solution of a nonlinear wave equation, which is taken to slowly vary so that its parameters, such as amplitude and width, are slowly varying functions of space (and time)[45, 78]. Slowly is used in relation to the frequency of the beams, the envelope may change over time but the change is slow. However, even the equations for a single nematicon have no known general solitary wave solution, only isolated solutions for fixed values of the parameters q and ν[74]. To overcome this lack of an exact solution on which to base modulation theory, suitable trial functions are chosen for this unknown solution[90, 91], which are then substituted into the Lagrangian formulation of the governing equations.

Lagrangian formulations can be used to find approximate evolution equations using trial functions based on the exact solutions of the NLS equation (1.3) [91]. The trial solution is now substituted into the Lagrangian and then averaging (integrating in x and y from $-\infty$ to ∞) results in an averaged Lagrangian based on the beam parameters which are functions of z. In the context of nonlinear beams in nematic liquid crystals, variational approximations have been found to give solutions in good agreement with experimental results[80, 92, 93] and numerical solutions[73, 76, 94, 105, 106, 107].

The trial functions used in the averaged Lagrangian are based on the soliton solution of the single focussing NLS equation (1.3), given in equation (1.4). The use of the sech function in the averaged Lagrangian results in interaction integrals that cannot be evaluated exactly so the trial functions used were based on the Gaussian function. The motivation for this is illustrated by the Figure 2.3. The parameters in the Gaussian trial function can be tuned so that the integration of the trial functions, sech and $\exp(-x^2/B^2)$, produce the same result. The big advantage of using Gaussian trial functions is that all integrals, including interaction integrals, in the averaged Lagrangian can be evaluated exactly. The Gaussian and sech profiles are both good approximations to the nematicon solution in different regions of the steady nematicon profile **??**. The Gaussian is a good match to the nematicon profile at its peak, while the sech profile gives a better representation of the nematicon closer to its tail **??**.

The trial functions used were

$$u = a_u e^{-\chi_u^2/w_u^2} e^{i\psi_u}, \tag{2.50}$$

$$v = a_v e^{-\chi_v^2/w_v^2} e^{i\psi_v}, \tag{2.51}$$

$$\theta = \alpha_u e^{-2\chi_u^2/\beta_u^2} + \alpha_v e^{-2\chi_v^2/\beta_v^2}, \tag{2.52}$$

where

$$\chi_u = \sqrt{(x-\xi_u)^2 + y^2}, \quad \chi_v = \sqrt{(x-\xi_v)^2 + y^2},$$
$$\psi_u = \sigma_u + V_u\left(x - \xi_u\right), \quad \psi_v = \sigma_v + V_v\left(x - \xi_v\right). \tag{2.53}$$

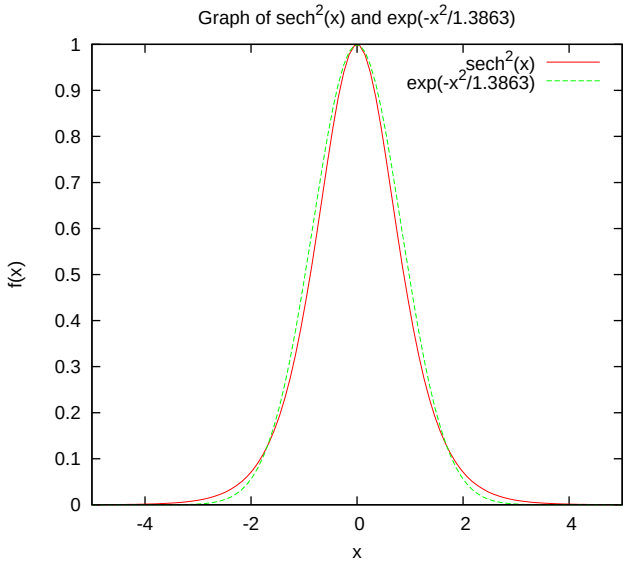

Figure 2.3: Plot of $\text{sech}^2(x)$ and $\exp(-x^2/B^2)$

The beam trial functions (2.50) and (2.51) are approximations to varying nematicons [73]. All the parameters of the trial functions (2.50)–(2.52) are usually taken to be functions of z [45, 73, 91]. However, it has been found that the amplitude-width oscillations in a_u, w_u and a_v, w_v nearly decouple from the position-velocity oscillations in ξ_u, V_u and ξ_v, V_v [59, 76, 77, 108, 109, 110, 111, 112]. The reduced modulation equations for the positions and velocities can then be obtained by assuming that the amplitudes a_u, a_v, α_u and α_v and widths w_u, w_v, β_u and β_v are constant, with only the positions ξ_u, ξ_v, velocities V_u, V_v and phases σ_u, σ_v depending on z. This approximation is the same as the particle approximation used for perturbed solitary wave theory [78].

The nematic equations (2.5) – (2.7) are non-dimensional equations, with all quantities being made non-dimensional using typical dimensional beam parameters[80, 81]. The beam widths are non-dimensionalised on a typical input width w_g of a Gaussian beam. A scale beam amplitude a_g is determined from the input power P_g of the Gaussian beam by[81]

$$P_g = \frac{\epsilon_0}{2} c n_o \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} |u_g|^2 \, dxdy = \frac{\epsilon_0}{2} c n_e \frac{\pi}{2} a_g^2 w_g^2, \tag{2.54}$$

where c is the speed of light in a vacuum and n_e is the extraordinary refractive index of the nematic medium. The non-dimensionalisation then depends on a typical input beam power, so that the parameters ν and q in the nematic equations (2.5) – (2.7) also depend on the input power. Of course, when the results are transformed back to dimensional variables, this dependence drops out[80, 81]. For the numerical results of the present paper, the non-dimensional amplitudes and widths of the input beams used were $a_u = a_v = 2.0$ and $w_u = w_v = 4.1$. For typical $\nu = O(100)$[80, 81], this gives that the ratios β_u/w_u and β_v/w_v are ~ 3, see equations (2.63) and (2.64) which determine β_u and β_v. We note that the amplitude and width of the director response are algebraically determined by the optical beam amplitudes and widths since the director equation (2.7) has no z derivatives. As the non-dimensional parameter ν controls the size of the ratios β_u/w_u and β_v/w_v, see (2.63) and (2.64), we refer to it as the "nonlocality," even though

experimentally it is the relative sizes of β_u and β_v to the beam widths that measure the nonlocal response. The final physical parameter of interest is the pre-tilt θ_0. The actual value of this parameter is not needed for the solution of the non-dimensional equations (2.5) – (2.7). It only arises when the non-dimensional quantities are converted back to dimensional variables, as it occurs in the definitions of ν and q [80, 81].

Substituting the trial functions into the Lagrangian (2.49) and averaging the Lagrangian, that is integrating in x and y from $-\infty$ to ∞[45], then results in an averaged Lagrangian whose variational equations are the modulation equations for its varying parameters as a function of the evolution variable z,

$$\mathcal{L} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} L dx dy. \tag{2.55}$$

This produces the averaged Lagrangian

$$\begin{aligned}\mathcal{L} &= -\pi a_u^2 w_u^2\left(\sigma_u' - V_u\xi_u'\right) - \pi a_v^2 w_v^2\left(\sigma_v' - V_v\xi_v'\right) - D_u\pi a_u^2 - D_v\pi a_v^2 \\ &\quad + 2A_u\pi a_u^2 w_u^2\alpha_u\beta_u^2 Q_1^{-1} - D_u\frac{\pi}{2}a_u^2 w_u^2 V_u^2 - D_v\frac{\pi}{2}a_v^2 w_v^2 V_v^2 - \pi\nu\alpha_u^2 - \pi\nu\alpha_v^2 \\ &\quad - \frac{\pi}{2}q\alpha_u^2\beta_u^2 - \frac{\pi}{2}q\alpha_v^2\beta_v^2 + 2A_v\pi a_v^2 w_v^2\alpha_v\beta_v^2 Q_3^{-1} - \phi,\end{aligned} \tag{2.56}$$

with the interaction potential ϕ as

$$\begin{aligned}\phi &= -2A_u\pi a_u^2 w_u^2\alpha_v\beta_v^2 Q_2^{-1}e^{-\gamma_1} - 2A_v\pi a_v^2 w_v^2\alpha_u\beta_u^2 Q_4^{-1}e^{-\gamma_2} \\ &\quad + 4\pi\nu\alpha_u\alpha_v\beta_u^2\beta_v^2 Q_5^{-2}\left[1-\gamma_3\right]e^{-\gamma_3} + 2\pi q\alpha_u\alpha_v\beta_u^2\beta_v^2 Q_5^{-1}e^{-\gamma_3}.\end{aligned} \tag{2.57}$$

In this averaged Lagrangian

$$\begin{aligned}&Q_1 = \beta_u^2 + w_u^2, \quad Q_2 = \beta_v^2 + w_u^2, \quad Q_3 = \beta_v^2 + w_v^2, \\ &Q_4 = \beta_u^2 + w_v^2, \quad Q_5 = \beta_u^2 + \beta_v^2, \\ &\gamma_1 = \frac{2\rho^2}{Q_2}, \quad \gamma_2 = \frac{2\rho^2}{Q_4}, \quad \gamma_3 = \frac{2\rho^2}{Q_5}.\end{aligned} \tag{2.58}$$

The distance ρ between the beams is

$$\rho = \xi_u - \xi_v. \tag{2.59}$$

This results in the variational, or modulation[45], equations

$$\begin{aligned}&\frac{1}{4}\left[\frac{d\sigma_u}{dz} - \frac{1}{2}V_u^2\right] = -\frac{1}{2}w_u^{-2} + \left[\alpha_u\beta_u^2\left(\beta_u^2 + w_u^2\right)Q_1^{-2}\right. \\ &\left.+\alpha_v\beta_v^2\left(\beta_v^2 + w_u^2\right)Q_2^{-2}e^{-\gamma_1} - \alpha_v w_u^2\beta_v^2\rho^2 Q_2^{-3}e^{-\gamma_1}\right], \qquad (2.60)\\ &\frac{1}{4}a_u^2 w_u^2\frac{dV_u}{dz} = -\frac{\partial\phi}{\partial\xi_u}, \qquad (2.61)\\ &\frac{d\xi_u}{dz} = D_u V_u, \qquad (2.62)\end{aligned}$$

plus the algebraic equations

$$\left(\nu + \frac{1}{2}q\beta_u^2\right)\alpha_u - \beta_u^2\left(a_u^2 w_u^2 Q_1^{-1} + a_v^2 w_v^2 Q_4^{-1} e^{-\gamma_2}\right) + q\alpha_v\beta_u^2\beta_v^2 Q_5^{-1} e^{-\gamma_3} + 2\nu\alpha_v\beta_u^2\beta_v^2 Q_5^{-2}\left(1 - 2\rho^2 Q_5^{-1}\right) e^{-\gamma_3} = 0, \tag{2.63}$$

$$q\alpha_u - 4A_u a_u^2 w_u^4 Q_1^{-2} - 4A_v a_v^2 w_v^4 Q_4^{-2} e^{-\gamma_2} - 8A_v a_v^2 w_v^2 \beta_u^2 \rho^2 Q_4^{-3} e^{-\gamma_2} + 8\nu\alpha_v\beta_v^2 Q_5^{-3}\left[\beta_v^2 - \beta_u^2 - \left(\beta_v^2 - 3\beta_u^2\right) 2\rho^2 Q_5^{-1} - 4\beta_u^2\rho^4 Q_5^{-2}\right] e^{-\gamma_3} + 4q\alpha_v\beta_v^2 Q_5^{-2}\left(\beta_v^2 + 2\beta_u^2\rho^2 Q_5^{-1}\right) e^{-\gamma_3} = 0, \tag{2.64}$$

Again, there are equivalent equations for the v beam.

2.7 Solving the modulation equations

The modulation equations produce a system of differential equations that can be represented as

$$\begin{aligned}
a_{11}\frac{du_1}{dz} + a_{12}\frac{du_2}{dz} + \cdots + a_{16}\frac{du_6}{dz} &= f_1(x_1, x_2, \ldots, x_6),\\
a_{21}\frac{du_1}{dz} + a_{22}\frac{du_2}{dz} + \cdots + a_{26}\frac{du_6}{dz} &= f_2(x_1, x_2, \ldots, x_6),\\
a_{31}\frac{du_1}{dz} + a_{32}\frac{du_2}{dz} + \cdots + a_{36}\frac{du_6}{dz} &= f_3(x_1, x_2, \ldots, x_6),\\
a_{41}\frac{du_1}{dz} + a_{42}\frac{du_2}{dz} + \cdots + a_{46}\frac{du_6}{dz} &= f_4(x_1, x_2, \ldots, x_6),\\
a_{51}\frac{du_1}{dz} + a_{52}\frac{du_2}{dz} + \cdots + a_{56}\frac{du_6}{dz} &= f_5(x_1, x_2, \ldots, x_6),\\
a_{61}\frac{du_1}{dz} + a_{62}\frac{du_2}{dz} + \cdots + a_{66}\frac{du_6}{dz} &= f_6(x_1, x_2, \ldots, x_6).
\end{aligned} \tag{2.65}$$

For the modulation equations shown in (2.60)–(2.62) the variables to solve for are σ_u, σ_v, V_u, V_v, ξ_u and ξ_v.

In vector form these equations can be written as

$$\mathbf{A}\mathbf{x}' = \mathbf{b}. \tag{2.66}$$

Here $\mathbf{A}$ is the matrix composed of the coefficients of $\mathbf{x}'$ evaluated by substituting the values of the parameters at the starting point for the step and $\mathbf{x}'$ is the column vector made up of the beam parameters that have not been calculated from the algebraic equations. The vector $\mathbf{b}$ is the vector of the inhomogeneous terms in each of the differential equations. The system (2.66) can be solved with a standard Runge-Kutta method. Theoretically, the system can be solved by inverting the matrix $\mathbf{A}$, but this is inefficient numerically. The matrix $\mathbf{A}$ can be factored into lower and upper triangular matrices to give

$$\mathbf{A}\mathbf{x}' = \mathbf{L}\mathbf{U}\mathbf{x}' = \mathbf{b}. \tag{2.67}$$

Setting $\mathbf{y} = \mathbf{U}\mathbf{x}'$, equation (2.67) can be replaced by the system of equations

$$\mathbf{L}\mathbf{y} = \mathbf{b}, \quad \mathbf{U}\mathbf{x}' = \mathbf{y}. \tag{2.68}$$

This system of equations can be solved easily. Numerical forward substitution is used to solve for $\mathbf{y}$ and then backward substitution solves $\mathbf{U}\mathbf{x}^{'} = \mathbf{y}$ for $\mathbf{x}^{'}$. The advantage of using $\mathbf{LU}$ factorisation is that there is no need to calculate the inverse matrix $\mathbf{A^{-1}}$ and the system (2.67) can be solved by forward and backward substitution.

The system (2.67) can now be solved using the standard fourth order Runge-Kutta scheme for $\mathbf{x}^{'}$ with the initial values of the beam parameters. The initial value problem is defined as

$$\mathbf{x}^{'} = \mathbf{A^{-1}b}, \quad \mathbf{x}(z_0) = \mathbf{x}_0, \tag{2.69}$$

where the procedure discussed above is used to solve $\mathbf{A^{-1}b}$.

The fourth order Runge-Kutta algorithm used to solve a system of differential equations is a simple generalisation of that for a single first order equation. Consider the system of equations

$$\begin{aligned}
\frac{du_1}{dz} &= f_1(x_1, x_2, \ldots, x_6),\\
\frac{du_2}{dz} &= f_2(x_1, x_2, \ldots, x_6),\\
\frac{du_3}{dz} &= f_3(x_1, x_2, \ldots, x_6),\\
\frac{du_4}{dz} &= f_4(x_1, x_2, \ldots, x_6),\\
\frac{du_5}{dz} &= f_5(x_1, x_2, \ldots, x_6),\\
\frac{du_6}{dz} &= f_6(x_1, x_2, \ldots, x_6).
\end{aligned} \tag{2.70}$$

The notation w_{ij} is used for the approximate solution $u_j(z)$ at the point z_j, with the step size h and initial conditions $w_{1,0} = \alpha_1$, $w_{2,0} = \alpha_2$,..., $w_{6,0} = \alpha_6$. The Runge-Kutta method for systems of differential equations is

$$k_{1,i} = hf_i(z_j, w_{1,j}, w_{2,j}, \ldots, w_{6,j}), \tag{2.71}$$
$$k_{2,i} = hf_i(z_j + h/2, w_{1,j} + k_{1,1}/2, w_{1,2} + k_{1,2}/2, \ldots, w_{6,j} + k_{1,6}/2), \tag{2.72}$$
$$k_{3,i} = hf_i(z_j + h/2, w_{1,j} + k_{2,1}/2, w_{1,2} + k_{2,2}/2, \ldots, w_{6,j} + k_{2,6}/2), \tag{2.73}$$
$$k_{4,i} = hf_i(z_j + h, w_{1,j} + k_{3,1}, w_{1,2} + k_{3,2}, \ldots, w_{6,j} + k_{3,6}). \tag{2.74}$$

Then

$$w_{i,j+1} = w_{i,j} + (k_{1,i} + 2k_{2,i} + 2k_{3,i} + k_{4,i})/6 \quad \text{for } i = 1, 2, \ldots, 6. \tag{2.75}$$

Note that each of $k_{1,1}$,$k_{1,2}$,...,$k_{1,6}$ must be calculated before $k_{2,1}$ can be determined.

2.8 Results

In this section numerical solutions of the two nematicon equations (2.5)–(2.7) will be compared with solutions of the modulation equations (2.60)–(2.62) and algebraic equations (2.63)–(2.64). The two nematicon equations were solved numerically using step sizes of $\Delta x = \Delta y = 0.4$ and $\Delta z = 0.02$. The spatial interval in the x and y directions is 102.4, which was divided into 256 intervals. Comparisons were taken for

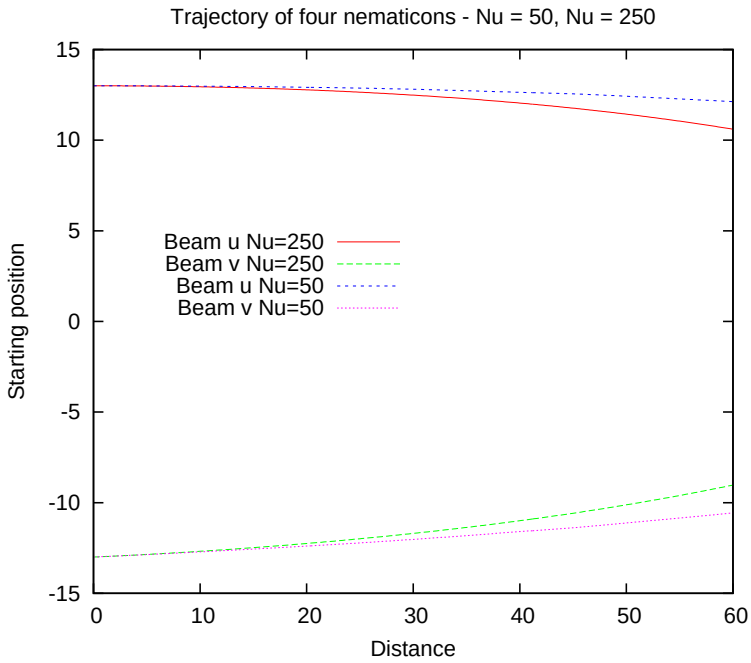

Figure 2.4: Paths of signal and control beams as given by full numerical solutions of nematic equations (2.1)–(2.3). Trajectory with $\nu = 250$: solid red line, u beam upper and dashed green line, v beam lower; trajectory with $\nu = 50$: dashed blue line, u beam upper and dashed purple line, v beam lower. The initial starting position of the signal beam is $\xi_{u0} = 13.0$ and the starting position of the control beam is $\xi_{v0} = -13.0$. There is no value set for the target ξ_{uf}. Here $V_{u0} = 0$, $V_v = 0.02634129$ radians and $L = 60$.

different step sizes and the values used produced a good compromise between accuracy and the time taken to produce the solutions. The main focus of this thesis is the trajectories taken by the optical paths, but other characteristics of the nematicons were important in determining the trajectories. The power of an optical beam has to be sufficient to form a nematicon and in turn the power is dependent on the amplitude and width of the beam. When the power is small, no nematicon can form and if the power is too large, then the nematicon can break up into two beams. To ensure the comparisons between the numerical solutions and the solutions from the modulation equations were valid, the parameters for all solutions have to be consistent.

The two nematicon equations (2.5)–(2.7) and the modulation equations (2.60)–(2.62) have been derived under the assumption that θ, the additional angle of rotation due to the optical beam, is small. This assumption is valid when the two nematicons are in a nonlocal medium. The degree of nonlocality of a nematic medium is measured by the parameter ν. The larger the value of ν, the higher is the degree of nonlocality and the greater the interaction between two optical beams. A typical value for ν in experimental work is $\nu \approx 200$ [49, 63]. Figure 2.4 shows the effect of ν on the interaction between two laser beams. Two scenarios have been run. In one the value of ν is 250, while in the other the value of ν is 50. When ν is set to 250, the width of the director is greater than for ν set to 50. This means there is more interaction between the beams and the attract each other to a greater extent. No target values have been set in either of the scenarios, however when ν is 250 the signal beam has reached the end of the cell at position 10.5 rather then the value of 12 when ν is 50.

Each time a solution is calculated, by whatever method, the optical beams are considered identical. The amplitudes are set to the same value, as are the widths of the beams. Typical values for the amplitude are $a = 2.0$ and for the width $w = 3.0$. If the beams had different wavelengths, then the values for the diffraction coefficients D_u and D_v would be different. The same is also true for the coupling coefficients, A_u and A_v. Values for β, the width of the director beam, and α, the amplitude of the director beam, are found as solutions of the algebraic equations. Typical values are $\beta \approx 12$, and $\alpha \approx 0.2$.

Figure 2.5 shows the paths taken by two beams, u and v. The starting position for beam u is $\xi_u = 13.0$ and $\xi_v = -13.0$ for beam v. Optical beam u is guided to the target value of 8.0 by changing the input angle of beam v to 0.0383259 radians. Once the target value is reached the input angle of v is recorded for comparison with the input angle of beam v calculated from the modulation equations to reach the same target value.

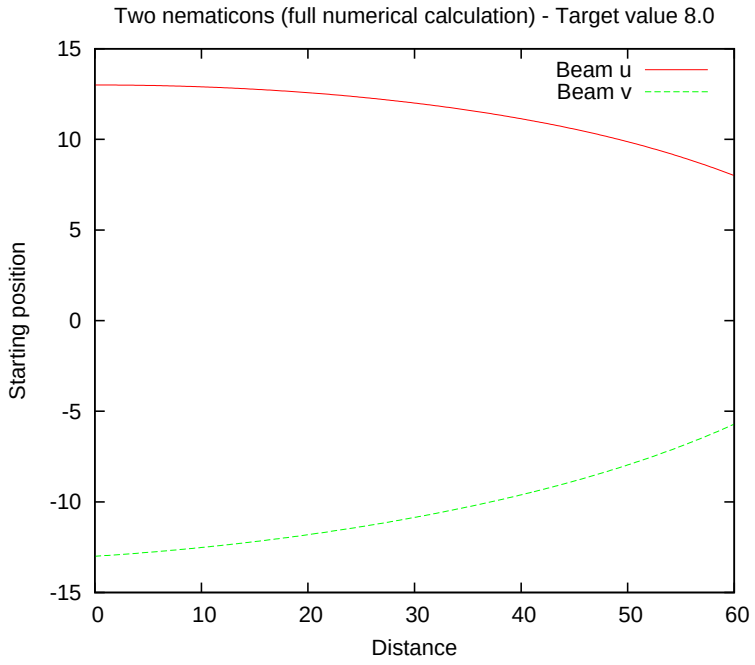

Figure 2.5: Beam u is guided to the target value, 8.0, by changing the input angle of beam v to 0.0383259 radians. Paths of signal and control beams as given by the solution of the two body particle model equations. The signal beam: solid (red) line; the control beam: dashed (green) line. The initial starting position of the signal beam is $\xi_{u0} = 13.0$ and the starting position of the control beam is $\xi_{v0} = -13.0$. The target position of the signal beam is $\xi_{uf} = 8.0$. Here $V_{u0} = 0$, $\nu = 225$ and $L = 60$.

Figure 2.6 shows the amplitude of the two beams. If the power of the two beams in not large enough, then the beams diffract and lose energy. The power of the beams is shown to be sufficient to cause the beams to focus and form nematicons since the amplitude of both beams is identical over the length of the cell and has not dissipated.

Another feature of interest for nematicons is the beam profile. In Figure 2.7 the shelf either side of the peak of the beams is small and indicates that there is little loss due to radiation. The carrier waves for the beams form a large potential well due to the large nonlocality and for any radiation loss to occur, the potential well must be

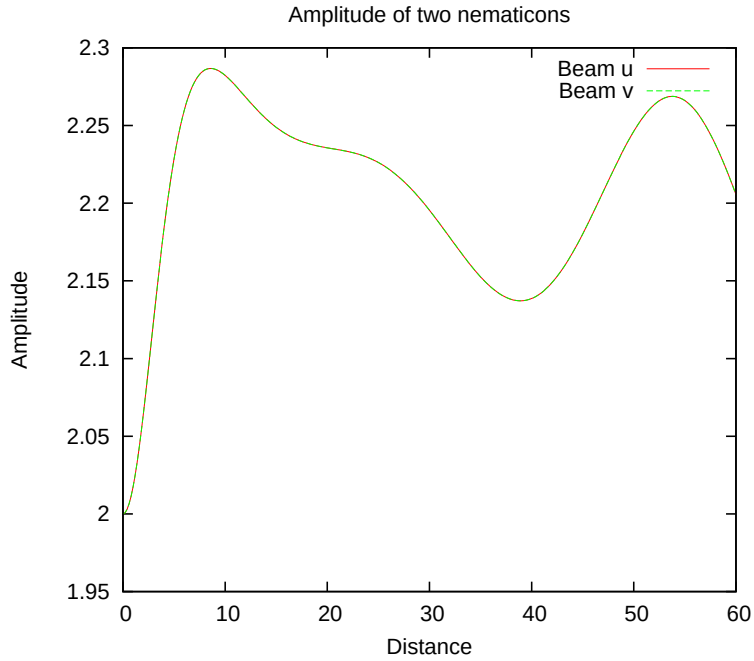

Figure 2.6: Amplitudes of beams u and v. The amplitudes are calculated for the signal and control beams from the full numerical solutions of the nematic equations (2.1)–(2.3). Amplitude for the signal beam: solid (red) line; amplitude for the control beam: dashed (green) line. The initial starting position of the signal beam is $\xi_{u0} = 13.0$ and the starting position of the control beam is $\xi_{v0} = -13.0$. The target position of the signal beam is $\xi_{uf} = 8.0$. Here the initial amplitude for both beams is 2.0, $V_{u0} = 0$, $\nu = 225$ and $L = 60$.

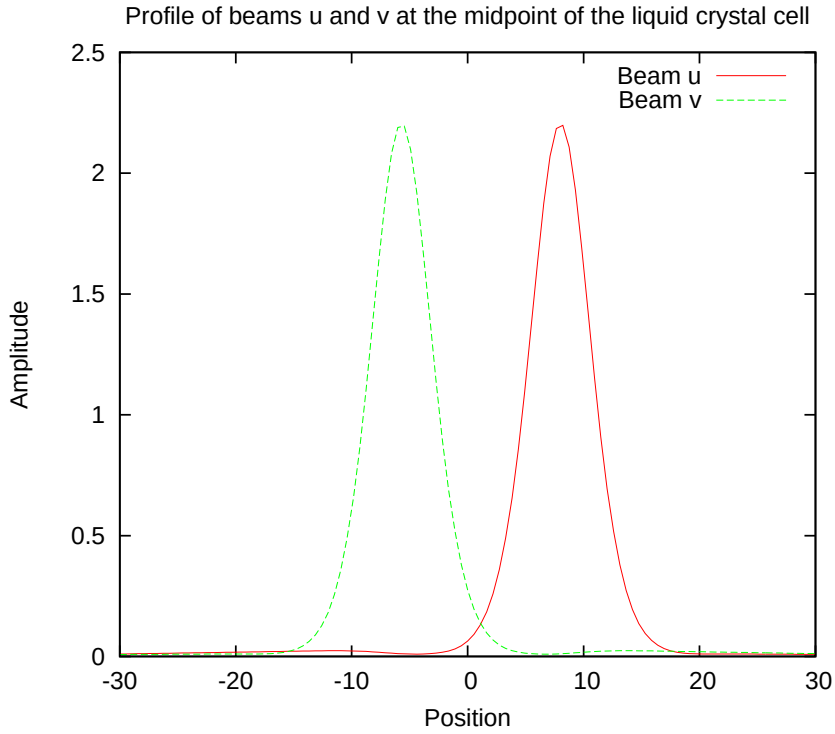

Figure 2.7: Profile for beams u and v with an initial Gaussian profile. The profiles are calculated for the signal and control beams from the full numerical solutions of the nematic equations (2.1)–(2.3). Profile for the signal beam: solid (red) line; profile for the control beam: dashed (green) line. The initial starting position of the signal beam is $\xi_{u0} = 13.0$ and the starting position of the control beam is $\xi_{v0} = -13.0$. The target position of the signal beam is $\xi_{uf} = 8.0$. Here $V_{u0} = 0$, $\nu = 225$ and $L = 60$.

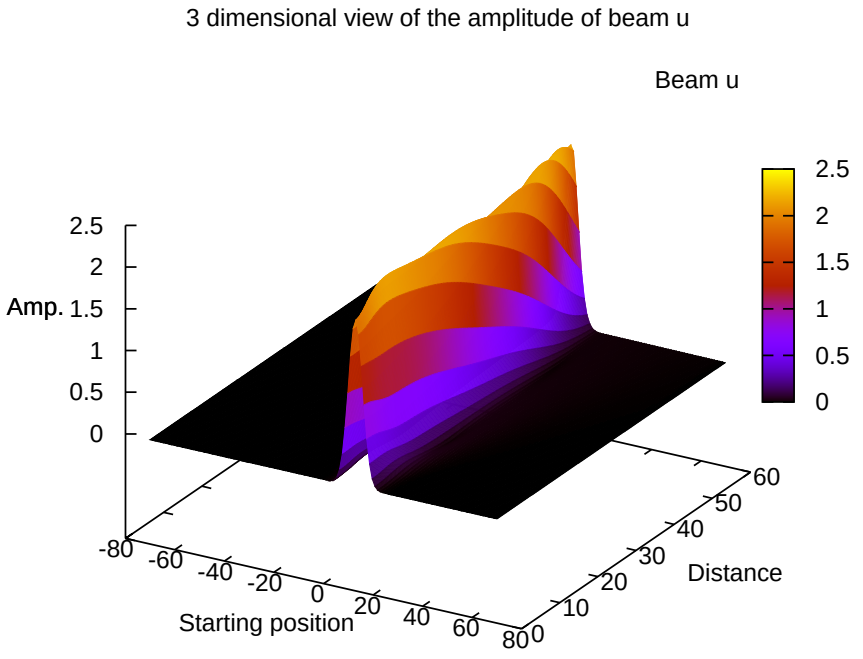

Figure 2.8: Three dimensional view for beam u. The initial starting position of the signal beam is $\xi_{u0} = 13.0$. The target positions of the signal beam are $\xi_{uf} = 8.0$. Here $V_{u0} = 0$, $\nu = 225$ and $L = 60$.

overcome.

So far the features of the nematicons have been shown in two dimensional form. Figure 2.8 shows that a nematicon has three dimensional features. It has a Gaussian profile in both the x and y dimensions and has features of a continuum or solid. Many of the figures in this thesis show the optical beams as two dimensional objects. However this is only a convenience in showing certain characteristics of the beams. It is important to grasp this in the context of Chapter 3.1 where it plays an important part in refining the potential between two nematicons.

A comparison of the trajectories calculated using the numerical procedure and from the modulation equations is shown in Figures 2.9 and 2.10. The points to note here are that the trajectories for the beam u calculated by the numerical procedure and from the modulation equations almost coincide. The starting point is the same and the target point is also the same, so there is very little freedom to deviate from the calculated path.

The other point to note is that path of beam v calculated from the modulation equations has a higher initial angle than the path calculated for beam v by the numerical procedure making the modulation v beams closer to the u beam. This is an indication that the attractive force between the beams calculated from the modulation equations is not as strong as the force calculated by the numerical procedure. The potential or attractive force between the beams is a function of the distance between the two beams. When the beams are close, the force between the two beams is strong and as the distance between the beams increases, then the force decreases. To guide the

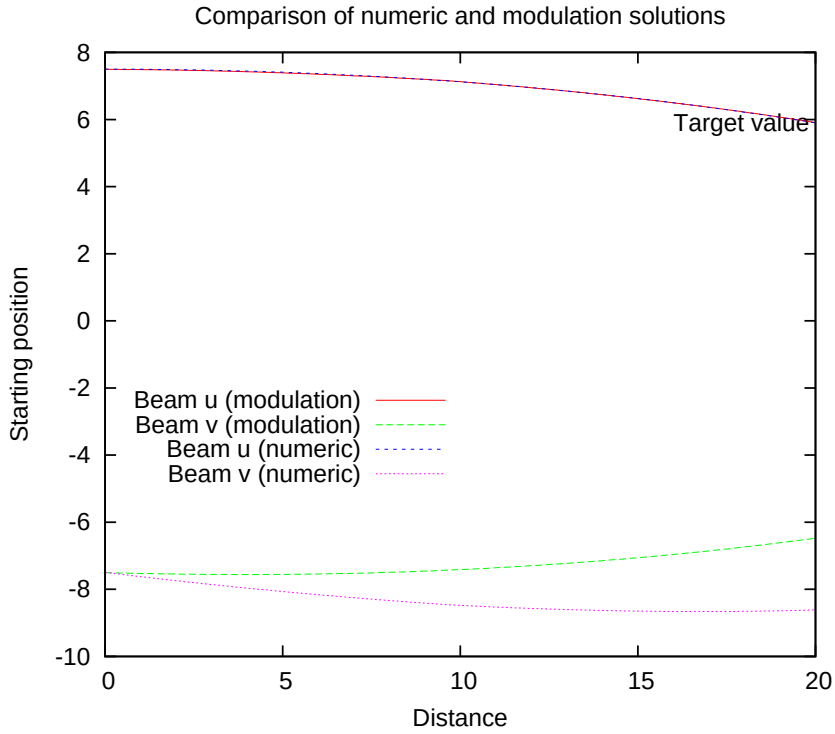

Figure 2.9: Paths of signal and control beams as given by full numerical solutions of nematic equations (2.1)–(2.3) and the two body particle model. Numerical trajectory: solid (red) line, u beam upper and v beam lower; two body particle model: dashed (green) line, u beam upper and v beam lower. The initial starting position of the signal beam is $\xi_{u0} = 7.8$ and the starting position of the control beam is $\xi_{v0} = -7.8$. The target position of the signal beam is $\xi_{uf} = 5.9$. Here $V_{u0} = 0$, $\nu = 225$ and $L = 20$.

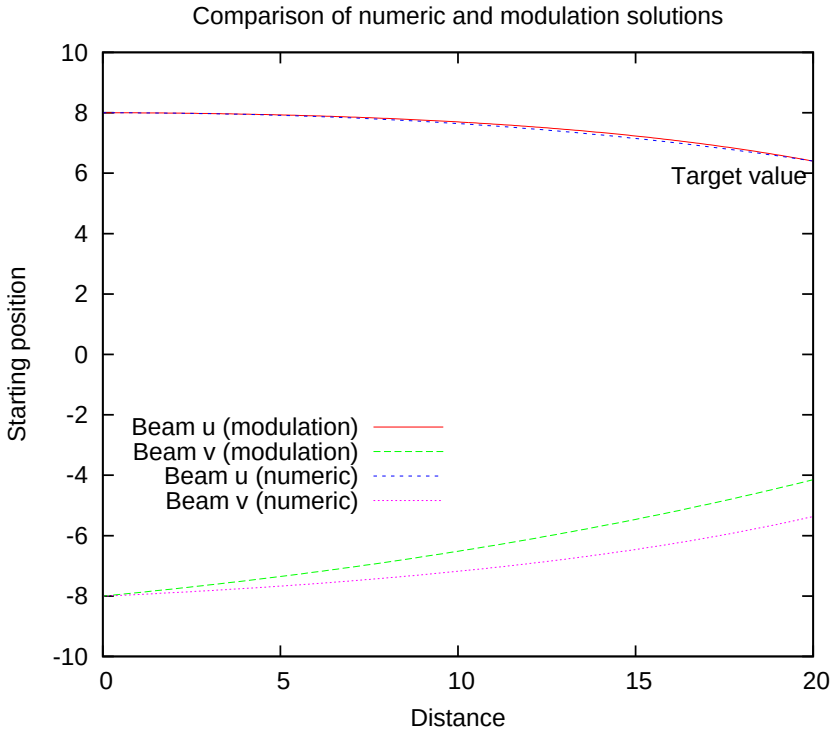

Figure 2.10: Paths of signal and control beams as given by full numerical solutions of nematic equations (2.1)–(2.3) and the two body particle model. Numerical trajectory: solid (red) line, u beam upper and v beam lower; two body particle model: dashed (green) line, u beam upper and v beam lower. The initial starting position of the signal beam is $\xi_{u0} = 8.0$ and the starting position of the control beam is $\xi_{v0} = -8.0$. The target position of the signal beam is $\xi_{uf} = 6.4$. Here $V_{u0} = 0$, $\nu = 225$ and $L = 20$.

u beam to the target value the v beam needs to be closer to the u beam using the particle model than would be predicted by the solution of the governing equations for two nematicons. Only two examples have been shown here describing the initial angle predicted by the modulation equations being greater than the angle predicted by the governing equations. However, this is true in general and in the Chapter 3.1, more extensive data is shown. In addition, an adjustment is made to the potential in the extended particle model that brings the initial angle of the v beam more in line with the angle predicted by the full governing equations.

Figure 2.11 shows the trajectories of the full numerical solutions of the nematic equations (2.1)–(2.3) and the trajectories of the solutions calculated for the modulation equations derived from the two body particle approximation. The parameters used for the results shown in this Figure are $\xi_{u0} = 13$, $\xi_{v0} = -13$, $\nu = 200$ and $L = 60$. The target figure has been varied for each of the three different cases. As expected the paths for the signal beam are close to each other. The starting position ξ_{u0} and the target position ξ_{uf} are the same for each result so the paths should be similar. The paths of the control beam, however show poor agreement. The control beam trajectory predicted by the two body particle approximation needs to initially begin at a higher angle than that for the full numerical solution because the attractive force between the two optical nematic beams is being underestimated by the two body particle approximation compared to the full numerical solution. The nematic beams using the two body particle approximation need to be closer to each other, so enough attractive force is exerted to pull the u beam to the target value.

Figure 2.12 shows the trajectories for optical beams calculated for regimes that are similar to those shown in Figure 2.11. The initial separation between the beams has been increased, so that ξ_{u0} is 16.0 and ξ_{v0} is -16.0. The nematic response of the liquid crystal cell has been increased by raising the value of the parameter ν to 225. The three cases shown have target values a) 13.0, b) 14.3 and c) 15.0. Again the initial angle for the control beam predicted by the two body particle approximation is higher than the initial angle of the control beam solution calculated from the full numerical solution of the coupled nematic equations (2.1)–(2.3). The potential between the optical beams calculated using the two body particle approximation is smaller than the potential between the beams calculated using the full numerical method. The beams need to get closer to each other to pull the signal beam to the target value. The initial separation of the nematic beams for the regimes shown in Figure 2.12 is greater than the regimes used in Figure 2.11. To overcome this increased initial separation the difference between the initial angles of the solutions calculated by the two body particle approximation and the full numerical method has also increased and Figure 2.12 shows more divergence between the paths than shown in Figure 2.11.

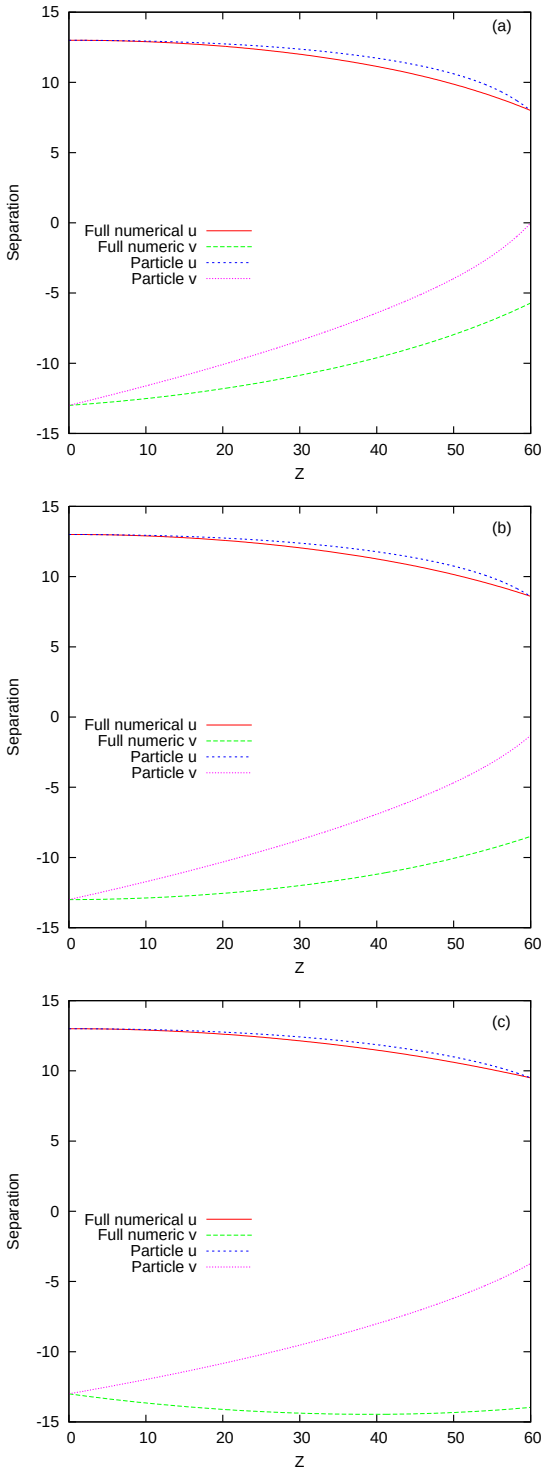

Figure 2.11: Paths of signal and control beams as given by full numerical solutions of nematic equations (2.1)–(2.3) and the two body particle approximation. Numerical trajectory: solid (red) line, u beam upper and dash (green) v beam lower; two body particle approximation: dashed (blue) line, u beam upper and dotted (purple) v beam lower. The target positions of the signal beam are (a) $\xi_{uf} = 8.0$, (b) $\xi_{uf} = 8.6$, (c) $\xi_{uf} = 9.5$. Here $\xi_{u0} = 13$, $\xi_{v0} = -13$, $\nu = 200$ and $L = 60$.

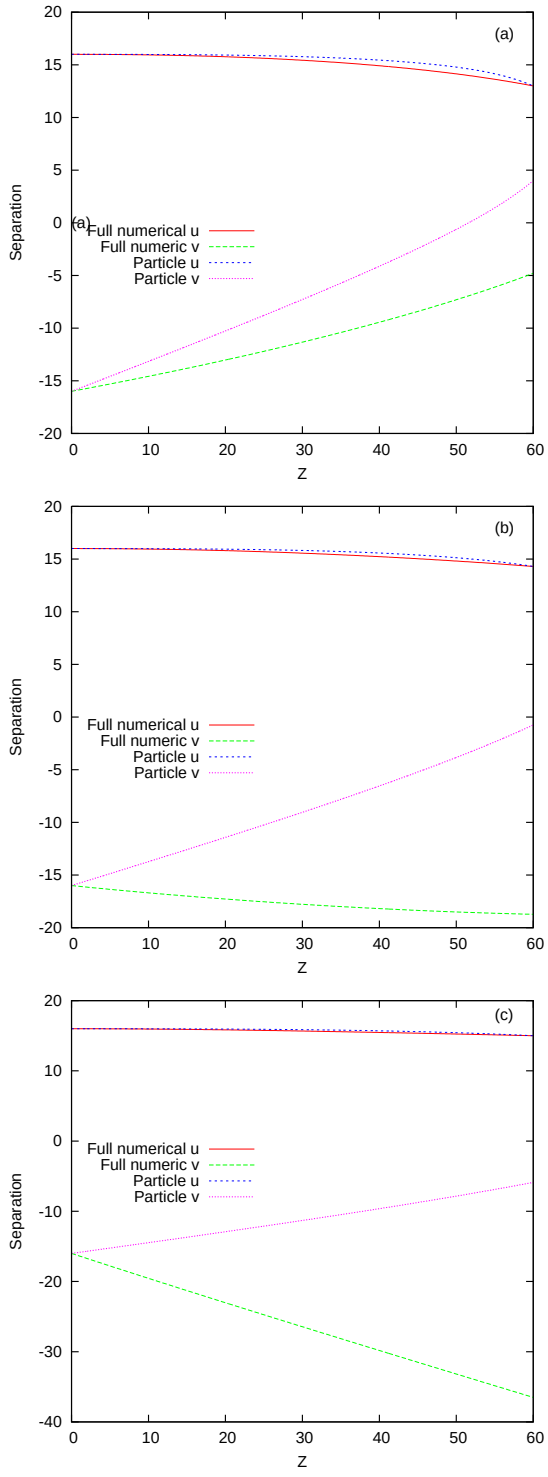

Figure 2.12: Paths of signal and control beams as given by full numerical solutions of nematic equations (2.1)–(2.3) and the two body particle approximation. Numerical trajectory: solid (red) line, u beam upper and dashed (green) line v beam lower; two body particle approximation: dashed (blue) line, u beam upper and dotted (purple) line v beam lower. The target positions of the signal beam are (a) $\xi_{uf} = 13.0$, (b) $\xi_{uf} = 14.3$, (c) $\xi_{uf} = 15.0$. Here $\xi_{u0} = 16$, $\xi_{v0} = -16$, $V_{u0} = 0$, $\nu = 225$ and $L = 60$.

Chapter 3

The extended particle model

3.1 Deriving the extended particle model

The behaviour of nematicons has many similarities with particles in a gravitational field and many of the ideas from classical mechanics can be used in the study of solitary waves in nematic liquid crystals. We want to take advantage of the knowledge that has been built up as part of classical mechanics in our research on nematicons. Before looking at the extended particle model we will derive some basic results from classical mechanics [119].

The work done on moving a particle from point 1 to point 2 is by definition

$$W_{12} = \int_1^2 \mathbf{F}\cdot d\mathbf{s}. \tag{3.1}$$

Here $\mathbf{F}$ is the force applied in moving the particle along the path $\mathbf{s}$. The particle has mass m and velocity v. If the mass of the particle is constant equation (3.1) reduces to

$$\int \mathbf{F}\cdot d\mathbf{s} = m\int \frac{d\mathbf{v}}{dt}\cdot\mathbf{v}dt = \frac{m}{2}\int \frac{d}{dt}(v^2)dt, \tag{3.2}$$

since $\mathbf{F} = \frac{d}{dt}(m\mathbf{v})$ and $\mathbf{v} = \frac{d\mathbf{s}}{dt}$ [119]. Therefore, the work done in moving the particle from point 1 to point 2 is also given by

$$W_{12} = \frac{m}{2}\left(v_2^2 - v_1^2\right). \tag{3.3}$$

The quantity $\frac{1}{2}mv^2$ is the kinetic energy and is denoted by T, so the work done is equal to the change in kinetic energy between the two points [119]

$$W_{12} = T_2 - T_1. \tag{3.4}$$

If the work done is the same for all paths between points 1 and 2, then the force is said to be conservative. A well known theorem in vector analysis [120] says a necessary and sufficient condition that W_{12} is independent of the path taken is that the force $\mathbf{F}$ is the

gradient of some scalar function of position, or

$$\mathbf{F} = -\nabla V(\mathbf{r}). \tag{3.5}$$

The scalar function of position $V(\mathbf{r})$ is called the potential or potential energy. In a conservative force field then [119]

$$W_{12} = \int \mathbf{F}\cdot d\mathbf{s} = -\int \nabla V(\mathbf{s})\cdot d\mathbf{s} = V_1 - V_2. \tag{3.6}$$

Combining equations (3.4) and (3.6) gives the result

$$T_2 - T_1 = V_1 - V_2 \quad \text{or} \quad T_1 + V_1 = T_2 + V_2. \tag{3.7}$$

This equation states that the total energy of a particle, made up of the sum of the kinetic and potential energies, is constant, or that energy is conserved. Using Newton's second law of motion with equation (3.5) results in the equation [119]

$$m\frac{d^2r}{dt^2} = -\frac{dV}{dr}. \tag{3.8}$$

This equation relates the mass and the acceleration of a particle with the gradient of the potential energy function. Using this equation allows us to describe the motion of a particle in a potential field $V(r)$. In the case of a planet revolving around the sun, the potential is given by the gravitational attraction between the planet and the sun. The masses involved are the mass of the planet and the mass of the sun. There is a connection between this equation 3.8 and the equations describing the motion of nematicons derived from the modulation equations in Chapter 2.

The equations (2.61) and (2.62) for the positions of the nematicons, plus their v beam equivalents, can be reduced to dynamical equations for two mechanical particles under a central force whose potential (ϕ) is given by equation (2.57) [78]. In this analogy, ξ_u and ξ_v are the positions of the particles, while V_u and V_v are their velocities. The modulation equations (2.61) and (2.62) and their v equivalents then reduce to the mechanical system

$$\begin{aligned} M_u\frac{d^2\xi_u}{dz^2} &= -\frac{\partial\phi}{\partial\xi_u} = -\frac{\partial\phi}{\partial\rho}, \\ M_v\frac{d^2\xi_v}{dz^2} &= -\frac{\partial\phi}{\partial\xi_v} = \frac{\partial\phi}{\partial\rho}. \end{aligned} \tag{3.9}$$

The "masses" of the nematicons are

$$M_u = \frac{1}{4}\frac{a_u^2 w_u^2}{D_u} \quad \text{and} \quad M_v = \frac{1}{4}\frac{a_v^2 w_v^2}{D_v}. \tag{3.10}$$

Comparing equations (3.9) with equation (3.8), we see that equations (3.9) are equivalent to the equation of motion of a particle of mass m in a potential field given by $V(r)$. For the nematicon equations the potential is the function ϕ defined by equation (2.57). Physically, the masses M_u and M_v are the optical powers of the two beams. Unfortunately, the potential ϕ, (2.57), is not proportional to the masses, as in Newto-

nian gravitation. As the masses M_u and M_v cannot be divided out, further analysis is greatly simplified if the restriction to beams of equal initial masses, i.e. equal optical powers, is taken, so that $M_u = M_v$.

For equal masses, the dynamical equations (3.9) have the centre of mass coordinate $\Xi_{cm} = \xi_u + \xi_v$. With this centre of mass coordinate and with $\rho = \xi_u - \xi_v$, the particle equations (3.9) then transform to

$$\frac{d^2\Xi_{cm}}{dz^2} = 0, \quad M_u\frac{d^2\rho}{dz^2} = -2\frac{\partial\phi}{\partial\rho}. \tag{3.11}$$

The centre of mass has constant velocity, as expected, with the separation of the beams given by the second of (3.11). The centre of mass equation can now be integrated to give the momentum conservation result

$$\xi_u + \xi_v = (V_{u_0} + V_{v_0})\, z + \xi_{u_0} + \xi_{v_0}, \tag{3.12}$$

since, as shown in Figure 2.1, the initial positions of the beams are $\xi_u = \xi_{u_0}$ and $\xi_v = \xi_{v_0}$ at $z = 0$. The separation equation, the second of (3.11), can be integrated to give the energy conservation equation

$$E = \frac{1}{2}M_u\dot{\rho}^2 + 2\phi(\rho), \tag{3.13}$$

with the energy E a constant. The separation of the beams is hence determined by

$$\int_{\xi_{u_0}-\xi_{v_0}}^{\rho} \frac{d\rho}{\sqrt{E - 2\phi(\rho)}} = \pm\sqrt{\frac{2}{M_u}}\, z. \tag{3.14}$$

Noting that the final positions of the beams are ξ_{u_f} and ξ_{v_f} at $z = L$ for the u and v beams, respectively, the momentum conservation result (3.12) gives the relation

$$V_{v_0} = \frac{1}{L}\left(\xi_{u_f} + \xi_{v_f} - \xi_{u_0} - \xi_{v_0}\right) - V_{u_0} \tag{3.15}$$

which links the positions of the input and output beams. In a similar manner, the energy conservation equation (3.14) gives another relation linking the input and output beams

$$\int_{\xi_{u_0}-\xi_{v_0}}^{\xi_{u_f}-\xi_{v_f}} \frac{d\rho}{\sqrt{E - 2\phi(\rho)}} = \pm\sqrt{\frac{2}{M_u}}\, L. \tag{3.16}$$

The $\pm$ sign is determined by whether $\xi_{u_f} - \xi_{v_f} > \xi_{u_0} - \xi_{v_0}$ or $\xi_{u_f} - \xi_{v_f} < \xi_{u_0} - \xi_{v_0}$. The two conservation equations (3.15) and (3.16) enable the input angle (velocity) V_{v_0} of the control beam v to be determined so that the signal beam u exits at a given ξ_{u_f}.

The modulation equations developed so far have assumed that the optical beams can be approximated as point particles [76]. However, as discussed in Section 2.8, the point particle modulation equations (3.9) give results which are in poor agreement with full numerical solutions of the nematic equations (2.1)–(2.3). This is because, as can be seen in Figures 3.5(a) and (b) and 3.8(a), that while the beams are initially well separated and have negligible overlap, especially in their tails, the beams can closely approach upon interaction. In these cases the point particle approximation

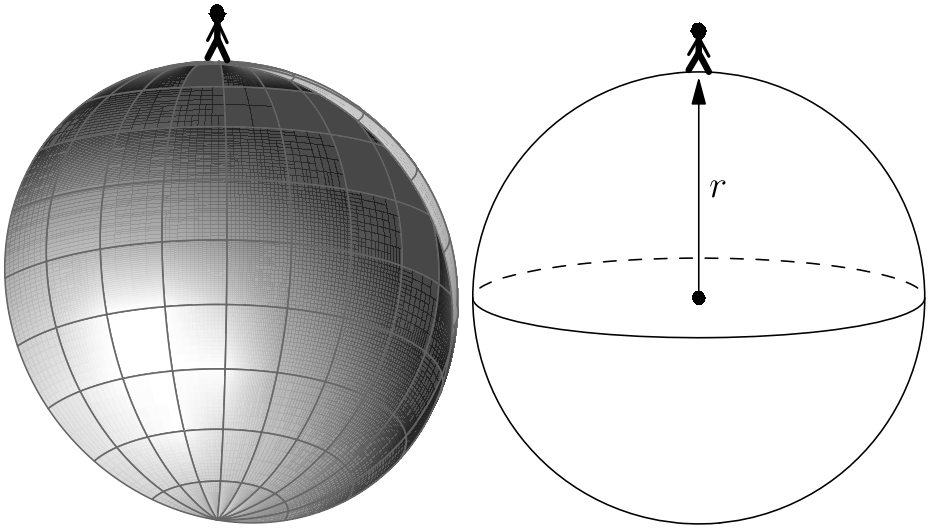

Figure 3.1: What is the weight of the man standing on the surface of the earth? Every part of the earth exerts some attraction on the man, so the weight of the man is the total contribution of all these forces of attraction. When the density of the earth is uniform and the force of attraction follows an inverse square law then the weight of the man can be determined from the image on the right. The mass of the earth can be thought of as concentrated at the centre.

underestimates the interaction between the nematicons and a better approximation is needed. This extra interaction in the tails was not important in previous studies of the interaction of nematicons as either the nematicons did not approach closely or the regions of closest approach were a minor part of their total interaction[58, 76, 77]. To obtain a more accurate estimate of the interaction potential between the beams their finite size must be taken into account.

Newton's law of gravity states that every object in the universe attracts every other object with a gravitational force proportional to the product of their masses and inversely proportional to the square of the distance between them

$$\text{Gravitational force} = G\frac{mM}{d^2}. \tag{3.17}$$

How do we determine the weight of a man standing on the surface of the earth? Every part of the earth is contributing something to the weight of the man (see the left hand side of Figure 3.1. Newton showed any spherically symmetric mass distribution can be treated as a point particle at its centre. When the density of the earth is uniform and since the force of attraction follows an inverse square law, then the force of gravity can be taken as if all the mass of the object were concentrated in a single point at the centre of mass of the earth [118]. In the case of the solar system where most of the objects of interest are to a good approximation a sphere and the law of gravitational attraction was an inverse square law, the difficulty was resolved in this way. This is shown in the right side of Figure 3.1. The problem has been reduced to a simpler problem and equation (3.17) can be safely applied.

In the analogy of the previous section the interaction of two optical beams and the interaction of two dynamical masses in a potential well can be further exploited to take

account of the finite size of the beams. This is done using ideas and methods dating back to Newtonian gravitation to find the gravitational potential of finite masses with arbitrary density distributions. It is known that when the gravitational potential follows an inverse square law, then spherical solids with uniform densities can be considered to be point particles with the mass concentrated at the centre [118]. In the case of optical beams the potential is Gaussian in form and the densities follow a Gaussian distribution, so some work is needed to find the centre of gravity. The centre of gravity cannot be assumed to be the centre of the Gaussian distribution (see top image of Figure 3.2). Let $\phi(x,y)$ be the potential for point masses and $P(x,y)$ be the density distribution of a general, finite mass. In the present analogy, $P(x,y)$ is the optical intensity distribution of a beam. Then the total potential Φ_{tot} due to the finite mass is given by

$$\Phi_{tot} \iint_{-\infty}^{\infty} P(x,y)\,dxdy = \iint_{-\infty}^{\infty} P(x,y)\phi(x,y)\,dxdy. \tag{3.18}$$

In the Newtonian gravitation analogy, P is the density distribution of the gravitating body and ϕ is the gravitational potential due to a point mass m_1, Gm_1/r. In the present case of optical beams the density is given by the trial functions (2.50) and (2.51), i.e. $\exp(-\chi_u^2/w_u^2)$ and $\exp(-\chi_v^2/w_v^2)$, and the potential is (2.57). The interaction potential between the two beams then contains three terms, (i) the interaction of a beam and the director distribution determined by the other beam (terms one and two in (2.57)), (ii) the interaction between the two parts of the director distribution forced by the two beams (term three containing the nonlocality ν in (2.57)), which is related to the nonlocal effect of the liquid crystal, and (iii) the interaction between the two parts of the director distribution (term four in (2.57)), which is related to the external applied field across the liquid crystal (i.e. q). Let us consider the contributions of each of these three terms in turn to the extended potential, taking account of the finite size of the optical beams.

For the first contribution the beam u profile is $\exp(-\chi_u^2/w_u^2)$ and from (2.57) the point potential between a point on the beam (X_u, Y_u) and a general point (X_v, Y_v) on the director is then (see Figure 3.2)

$$e^{-\frac{2[(X_u-X_v)^2+(Y_u-Y_v)^2]}{\beta_v^2+w_u^2}}. \tag{3.19}$$

We note that the width of this potential contribution has two contributions, the beam width w_u and the director response width β_v under the control beam. It was noted in Section 2.6 that $\beta_v \gg w_u$ as ν is large, $O(100)$, and the nematic response is highly nonlocal. This term could then be approximated by β_v^2 alone. However, the width contribution w_u^2 will be kept as the term (3.19) is then exact. The extended potential expression (3.18) therefore gives the corresponding potential contribution from a general point on the u beam over the entire width of the director distribution due to the other beam v as

$$\frac{w_u^2+\beta_v^2}{w_u^2+2\beta_v^2}\, e^{-\frac{2[(X_u-\xi_v)^2+(Y_u-\eta_v)^2]}{w_u^2+2\beta_v^2}}. \tag{3.20}$$

To calculate the effect of the finite width of the beam u the potential (3.18) is again used, but this time integrating over all points of the u beam. Taking a general point (X_u, Y_u) on the beam, the total potential contribution due to the finite beam u and the

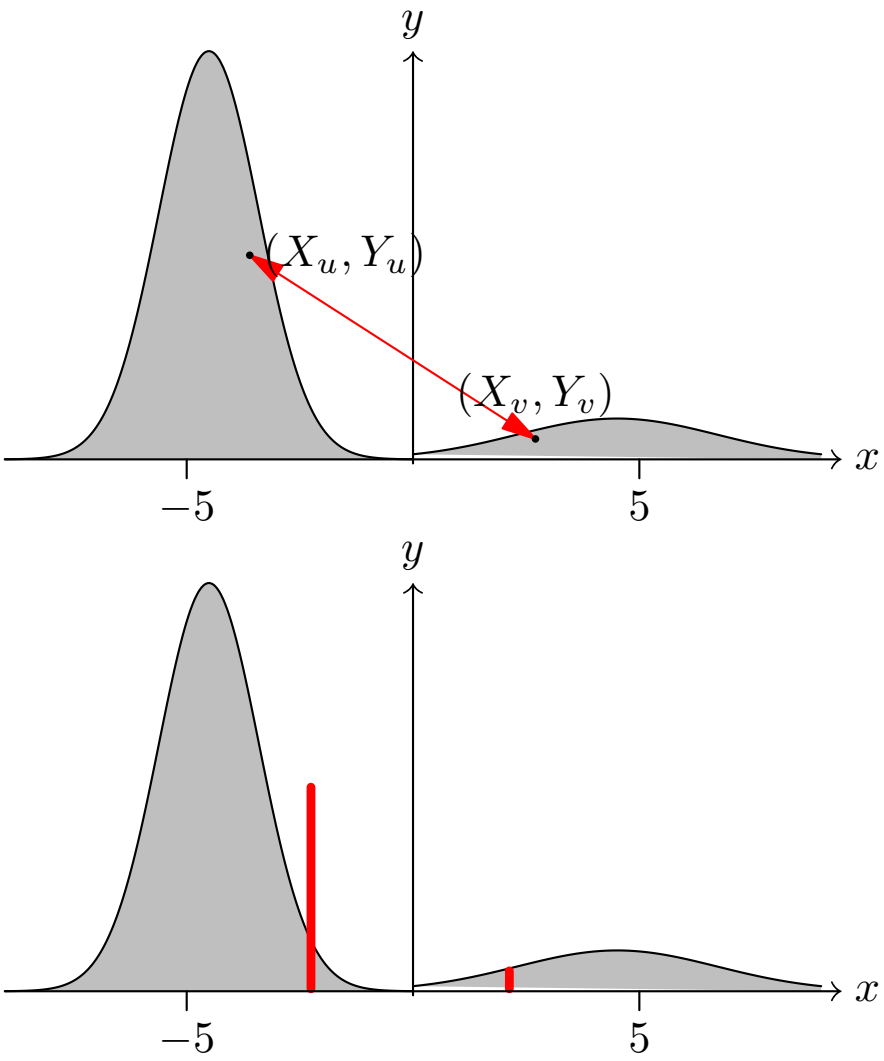

Figure 3.2: What is the total force of attraction between the two nematicons shown in the top figure? The force of attraction between the two nematicons is Gaussian in form and the profiles of the nematicons are also Gaussian. The centre of gravity cannot be assumed to be the centre of mass of the Gaussian profile. After carrying out the appropriate calculations the two nematicons can be considered to be equivalent to the two point masses shown in red. In effect the attractive force between the nematicons is stronger than the centre of the Gaussian profiles would suggest.

finite director response v is

$$\Phi_{(i)} = \frac{w_u^2+\beta_v^2}{w_u^2+2\beta_v^2}\frac{\iint_{-\infty}^{\infty} e^{-\frac{2[(X_u-\xi_v)^2+(Y_u-\eta_v)^2]}{w_u^2+2\beta_v^2}} e^{-\frac{(X_u-\xi_u)^2+(Y_u-\eta_u)^2}{w_u^2}} dX_u dY_u}{\iint_{-\infty}^{\infty} e^{-\frac{(X_u-\xi_u)^2+(Y_u-\eta_u)^2}{w_u^2}} dX_u dY_u}, \tag{3.21}$$

giving the result

$$\Phi_{(i)}(\xi_u,\xi_v) = \frac{w_u^2+\beta_v^2}{3w_u^2+2\beta_v^2} e^{-\frac{2\rho^2}{3w_u^2+2\beta_v^2}}. \tag{3.22}$$

See Figure 3.2.

The same reasoning can be applied to the interaction contributions (ii) and (iii), resulting in the separate potential contributions

$$\Phi_{(ii)}(\xi_u,\xi_v) = \frac{1}{4}\left(1-\frac{\rho^2}{\beta_u^2+\beta_v^2}\right) e^{-\frac{\rho^2}{\beta_u^2+\beta_v^2}}, \qquad (\nu) \tag{3.23}$$

$$\Phi_{(iii)}(\xi_u,\xi_v) = \frac{1}{2} e^{-\frac{\rho^2}{\beta_u^2+\beta_v^2}}. \qquad (q) \tag{3.24}$$

Adding together the potential contributions (3.22)–(3.24) gives the total potential on treating the beams as extended objects as

$$\begin{aligned}\Phi &= -2\pi A_u a_u^2 w_u^2 \alpha_v \beta_v^2 Q_6^{-1} e^{-\gamma_5} - 2\pi A_v a_v^2 w_v^2 \alpha_u \beta_u^2 Q_7^{-1} e^{-\gamma_6} \\ &\quad + \pi\nu\alpha_u\alpha_v\beta_u^2\beta_v^2 Q_5^{-2}\left[1-\gamma_4\right] e^{-\gamma_4} + \pi q \alpha_u\alpha_v\beta_u^2\beta_v^2 Q_5^{-1} e^{-\gamma_4}.\end{aligned} \tag{3.25}$$

Here

$$Q_6 = 2\beta_v^2+3w_u^2, \quad Q_7 = 2\beta_u^2+3w_v^2,, \quad \gamma_4 = \frac{\rho^2}{Q_5}, \quad \gamma_5 = \frac{2\rho^2}{Q_6}, \quad \gamma_6 = \frac{\rho^2}{Q_7}.$$

The approximate equations for the trajectories of the beams based on this new, extended potential are (3.9) with ϕ replaced by Φ.

A comparison of the functions Φ and ϕ is shown in table 3.1. For the function Φ the values in the Gaussian part of the function are lower than those in the function ϕ because the effective widths of the beams and director have increased. This is offset by the fractions used to multiply the Gaussian part of the functions. Using typical values of $w_u = 3.0$ and $\beta_v = 12.0$ the different parts of the functions have been plotted against ρ. The plots are shown in Figure 3.3 (a) to (c). In each of the plots the function ϕ has a higher value than Φ when the beams are close together, with the opposite occurring when the beams are far apart. Referring to Figures 3.5, 3.6 and 3.8 we see that the beams are rarely close together for the simulations that were carried out, so the attractive force calculated by the function Φ is higher than that calculated by ϕ for the simulations carried out here.

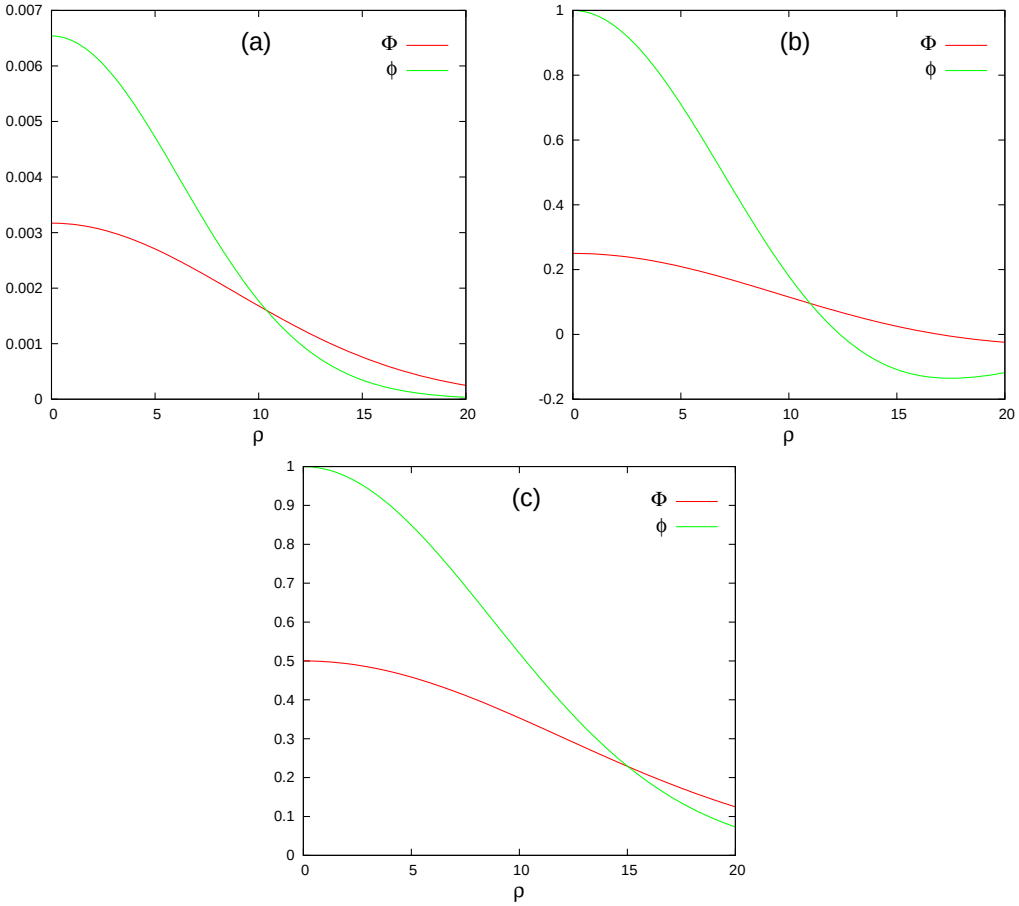

Figure 3.3: This figure shows a comparison of the extended potential Φ with the two body particle potential ϕ. Each graph shows the value of the potential versus ρ the distance between the beams. Plot (a) is for term (i) in the functions Φ and ϕ given by equations (3.25) and (2.57). Plots (b) and (c) are for terms (ii) and (iii). The graphs show that Φ is higher than ϕ for values of ρ greater than about 10 while the opposite is true for values of ρ less than 10. The function Φ produced higher values when the two beams were far apart. In plotting these functions typical values of $w_u = 3.0$ and $\beta_v = 12.0$ were used.

Term	$\Phi(\xi_u, \xi_v)$	$\phi(\xi_u, \xi_v)$
i	$\frac{1}{3w_u^2+2\beta_v^2} \exp\left(-\frac{2\rho^2}{3w_u^2+2\beta_v^2}\right)$	$\frac{1}{w_u^2+\beta_v^2} \exp\left(-\frac{2\rho^2}{w_u^2+\beta_v^2}\right)$
ii	$0.25\left(1-\frac{\rho^2}{\beta_u^2+\beta_v^2}\right) \exp\left(-\frac{\rho^2}{\beta_u^2+\beta_v^2}\right)$	$\left(1-\frac{2\rho^2}{\beta_u^2+\beta_v^2}\right) \exp\left(-\frac{2\rho^2}{\beta_u^2+\beta_v^2}\right)$
iii	$0.5 \exp\left(-\frac{\rho^2}{\beta_u^2+\beta_v^2}\right)$	$\exp\left(-\frac{2\rho^2}{\beta_u^2+\beta_v^2}\right)$

Table 3.1: Comparison of terms within Φ and ϕ. The value of Φ is higher than the value of ϕ when the beams are separated by a distance of more than 10. See Figure 3.3 for a detailed comparison of the values of Φ and ϕ. Typical values used for the width of the beam was $w_u = 3.0$ and for the width of the director $\beta_v = 12.0$.

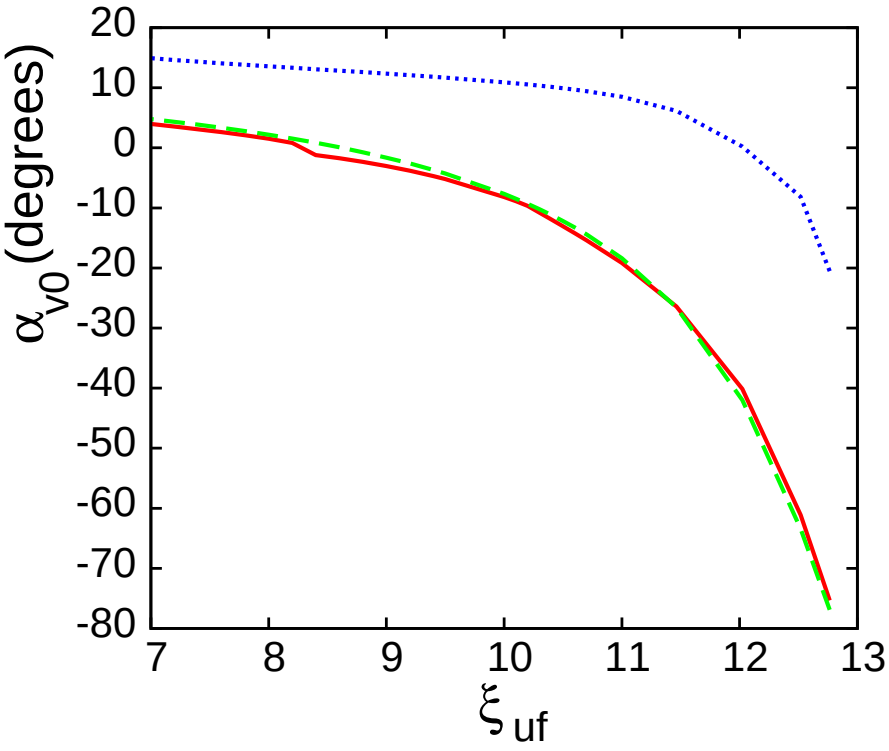

Figure 3.4: Input angle α_{v0} of control beam v needed to guide signal beam u to output position ξ_{uf}. The initial positions of the signal beam and the control beams are $\xi_{u0} = 13.0$ and $\xi_{v0} = -13.0$, respectively. Full numerical solution: dashed (green) line; particle approximation: dotted (blue) line; extended particle approximation: solid (red) line. Here $V_u = 0$, $\nu = 200$ and $L = 60$.

3.2 Solving the extended particle model equations

The utility of the particle and extended particle modulation equations of Sections 2.6 and 3.1, respectively, will be demonstrated by comparing their predictions for the control parameter, the input angle V_{v_0} of the control beam v, so that the signal beam exits at a given point ξ_{u_f}, with the predictions of these two sets of equations with full numerical solutions of the nematicon equations (2.1)–(2.3). The modulation equations (3.9) of Sections 2.6 and 3.1 were solved numerically using the standard 4th order Runge-Kutta scheme[114]. The NLS-type beam equations (2.1) and (2.2) were solved numerically using a pseudo-spectral method based on that of Fornberg and Whitham[115]. The director equation (2.3) was solved using a FFT based iterative method[76, 114].

From the particle equations (3.9) and using the potential for beams treated as extended objects (3.25), the extended particle equations for the trajectories of the beams are

$$M_u \frac{d^2\xi_u}{dz^2} = -\frac{\partial \Phi(\xi_u - \xi_v)}{\partial \xi_u}, \tag{3.26}$$

$$M_v \frac{d^2\xi_v}{dz^2} = -\frac{\partial \Phi(\xi_u - \xi_v)}{\partial \xi_v}, \tag{3.27}$$

where $M_u = \frac{a_u^2 w_u^2}{4D_u}$ and $M_v = \frac{a_v^2 w_v^2}{4D_v}$. Defining the variables $X = \xi_u - \xi_v$, $Y = \xi_u + \xi_v$ and assuming $M_u = M_v = M$, we have

$$M \frac{d^2\xi_u}{dz^2} = -\frac{d\Phi(X)}{dX} \quad \text{and} \tag{3.28}$$

$$M \frac{d^2\xi_v}{dz^2} = \frac{d\Phi(X)}{dX} \tag{3.29}$$

Adding equation (3.28) and (3.29) gives

$$M \frac{d^2 Y}{dz^2} = 0, \tag{3.30}$$

while subtracting equation (3.29) from equation (3.28) gives

$$M \frac{d^2 X}{dz^2} = -2\frac{d\Phi(X)}{dX}. \tag{3.31}$$

The solution of equation (3.30) is

$$Y = cz + d, \tag{3.32}$$

which is momentum conservation. So,

$$\xi_u + \xi_v = cz + d \quad \text{and} \tag{3.33}$$

$$\dot{\xi_u} + \dot{\xi_v} = V_{u_0} + V_{v_0} = c \tag{3.34}$$

From equation (3.31) the conservation of energy equation is derived

$$M\ddot{X} = -2\Phi_X, \tag{3.35}$$
$$M\dot{X}\ddot{X} = -2\Phi_X\dot{X}. \tag{3.36}$$

Integrating with respect to z gives

$$\frac{1}{2}M\dot{X}^2 = -2\int \frac{\partial\Phi(X)}{\partial X}\cdot\frac{dX}{dz}\cdot dz \tag{3.37}$$
$$= -2\Phi(X) + const \quad \text{or} \tag{3.38}$$
$$E = \frac{1}{2}M\dot{X}^2 + 2\Phi(X). \tag{3.39}$$

Hence the separation of the beams is given by

$$\dot{X} = \pm\sqrt{\frac{2}{M}(E - 2\Phi(X))}. \tag{3.40}$$

Substituting the initial conditions into equation (3.33) gives

$$\xi_{u0} + \xi_{v0} = d. \tag{3.41}$$

Some of the variables needed to calculate $\Phi(X)$ are not available as initial conditions and are obtained as solutions of the algebraic equations that are part of the modulation equations, (2.63)–(2.64), used to derive equations (3.26) – (3.27). The algebraic equations are solved to obtain the variables α_u, α_v, β_u and β_v.

3.3 Solving the modulation equations

The algebraic equations, (2.63)–(2.64) are of the form

$$\begin{aligned} F_1(\alpha_u, \alpha_v, \beta_u, \beta_v) &= 0, \\ F_2(\alpha_u, \alpha_v, \beta_u, \beta_v) &= 0, \\ F_3(\alpha_u, \alpha_v, \beta_u, \beta_v) &= 0, \\ F_4(\alpha_u, \alpha_v, \beta_u, \beta_v) &= 0, \end{aligned} \tag{3.42}$$

or in vector form

$$\mathbf{F}(\mathbf{x}) = \mathbf{0}, \tag{3.43}$$

where $\mathbf{F}$, $\mathbf{x}$, $\mathbf{0}$ are all vectors. Each of the functions, F_i for $i = 1, \ldots, 4$, is nonlinear and the solutions may not be unique. The method chosen to solve these equations was to first obtain an initial guess that was close to the solution of the equations (3.42). The initial guess was then used in a Newton-Raphson iteration to refine that guess and reach the desired degree of accuracy.

The method of Steepest Descent is used to obtain the initial approximation. Steepest descent was chosen because it is a robust way to obtain an initial guess that is reasonably close to the solution. The convergence of this method is first order, so it is more efficient to then pass the initial guess onto a faster converging method (such as

Newton-Raphson).

The method of steepest descent forms a function G made up from the system of equations that need to be solved [117]. The function G is then minimised to produce the initial guess. $G(x)$ is defined as

$$G(x) = F_1(x)^2 + F_2(x)^2 + F_3(x)^2 + F_4(x)^2, \tag{3.44}$$

where the functions $F_1(x)$ etc. are the functions that are part of the system of equations (3.42). From an initial point, a local minimum in $G(x)$ will be in the direction of $-\nabla(G) = (\frac{\partial G}{\partial \alpha_u}, \dots)$. The greatest decrease in the value of G(x) is in the direction of $-\nabla G(x)$, so to minimise $G(x)$ we travel in the direction of $-\nabla G(x)$ to reach a value lower than the starting value. We evaluate $-\nabla G(x)$ at the point, producing the lower value of $G(x)$ and repeat the process until the value of $G(x)$ is no longer lowered. We use the notation $x^{(0)}$ for the initial point used to start the steepest descent process and $x^{(i)}$ for the result after the ith iteration. The equation for the first iteration is [117]

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} - \alpha \nabla G(\mathbf{x}^{(0)}). \tag{3.45}$$

We now find a value of α so that $G(x^{(1)})$ is significantly less than $G(x^{(0)})$. The single valued function $h(\alpha)$ is defined by

$$h(\alpha) = G(\mathbf{x}^{(0)} - \alpha \nabla G(\mathbf{x}^{(0)})). \tag{3.46}$$

The value of α that minimises the function $h(\alpha)$ is the value of α needed for equation (3.45).

Finding a minimum value for $h(\alpha)$ directly is too costly in terms of computational time [117], so $h(\alpha)$ is defined as the quadratic polynomial that interpolates three points α_1, α_2 and α_3 that are hopefully close to the minimum. Defining the unit vector $\mathbf{z}^*$ as

$$\mathbf{z}^* = \frac{\nabla G(\mathbf{x})}{\|\mathbf{z}\|} \tag{3.47}$$

and the value of function $g_i(x)$ as

$$g_i = G(\mathbf{x} - \alpha_i \mathbf{z}^*), \tag{3.48}$$

allows the polynomial $P(\alpha)$ to be defined. The quadratic polynomial through the points α_1, α_2 and α_3 is

$$P(\alpha) = g_1 + \frac{g_2 - g_1}{\alpha_2} + \frac{1}{\alpha_3}\left(\frac{g_3 - g_2}{\alpha_3 - \alpha_2} - \frac{g_2 - g_1}{\alpha_2}\right)\alpha(\alpha - \alpha_2). \tag{3.49}$$

Setting α_1 to zero for computational convenience and then repeatedly halving α_3 from an initial value of 1, until we obtain the result $g_3 < g_1$ [117] gives the variable α_2, which can then be set to $\alpha_2 = \alpha_3/2$. The minimum value of the quadratic polynomial is either at the point α_3 or the point

$$0.5\left(\alpha_2 - \frac{(g_3 - g_2)\alpha_3}{\alpha_2} \Big/ \left(\frac{g_3 - g_2}{\alpha_3 - \alpha_2} - \frac{g_2 - g_1}{\alpha_2}\right)\right). \tag{3.50}$$

This procedure is iterated until the minimum value of $G(x)$ is reached within a set level of tolerance.

Once a sufficiently accurate initial approximation has been found, this can be passed to the Newton-Raphson method and this method gives much faster convergence. The Newton-Raphson method is based on the recurrence relation

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \mathbf{J}(\mathbf{x}^{(k)})^{-1}\mathbf{F}(\mathbf{x}^{(k)}), \tag{3.51}$$

where $\mathbf{J}$ is the Jacobian matrix defined as

$$\mathbf{J}(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1(\mathbf{x})}{\partial \alpha_u} & \frac{\partial f_1(\mathbf{x})}{\partial \alpha_v} & \cdots & \frac{\partial f_1(\mathbf{x})}{\partial \beta_v} \\ \frac{\partial f_1(\mathbf{x})}{\partial \alpha_u} & \frac{\partial f_1(\mathbf{x})}{\partial \alpha_v} & \cdots & \frac{\partial f_1(\mathbf{x})}{\partial \beta_v} \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_1(\mathbf{x})}{\partial \alpha_u} & \frac{\partial f_1(\mathbf{x})}{\partial \alpha_v} & \cdots & \frac{\partial f_1(\mathbf{x})}{\partial \beta_v} \end{bmatrix}. \tag{3.52}$$

$\mathbf{F}(\mathbf{x})$ is the coordinate matrix defined by equation (3.42) and $\mathbf{x}^{(k)}$ is the starting approximation used in the $(k-1)$th iteration. The matrix $\mathbf{J}(\mathbf{x}^{(k)})^{-1}$ is not evaluated for efficiency reasons. However, the vector $\mathbf{y}$ is found that satisfies the equation

$$\mathbf{J}(\mathbf{x}^{(k)})\mathbf{y} = -\mathbf{F}(\mathbf{x}^{(k)}). \tag{3.53}$$

Equation (3.53) is a $n \times n$ system of equations that is solved using Gaussian elimination to obtain $\mathbf{y}$. Using equation (3.51), the vector $\mathbf{y}$ is then added to the vector $\mathbf{x}^{(k)}$ to obtain the next approximation for the iteration. The iteration is stopped when the norm of the vector $\mathbf{y}$ is less than the specified tolerance. The values of α_u, α_v, β_u and β_v are now available for use in solving equation (3.40).

The sign of equation (3.40) is initially determined from the sign of $V_{u_0} - V_{v_0}$. If this quantity is positive, then the positive square root is taken. If it is negative, then the negative square root is taken. The value of E is the total energy of the system and is a constant. The value of E is calculated from the initial kinetic and potential energies as

$$E = \text{Kinetic energy} + \text{Potential energy} \tag{3.54}$$

$$= \frac{1}{2}M_u\dot{X}^2 + \Phi(\xi_u, \xi_v) \tag{3.55}$$

$$= \frac{1}{2}M_u(V_{u_0} + V_{v_0})^2 + \Phi(\xi_{u_0}, \xi_{v_0}). \tag{3.56}$$

Equation (3.40) is a first order ordinary differential equation in the variable X with an initial condition. This can be solved using a standard fourth order Runge-Kutta method [117]. We have the differential equation

$$\frac{dX}{dz} = f(X), \quad f(X) = \pm\sqrt{\frac{2}{M}(E - 2\Phi(X))}, \tag{3.57}$$

where the initial condition is $X = X_0$ at $z = 0$. Defining w_j as the approximation for X at time z_j and with time step h, the fourth order Runge-Kutta method first calculates

the intermediate steps

$$k_1 = hf(w_j, w_2), \tag{3.58}$$
$$k_2 = hf(w_j + k_1/2), \tag{3.59}$$
$$k_3 = hf(w_j + k_2/2), \tag{3.60}$$
$$k_4 = hf(w_j + k_3) \tag{3.61}$$

and the approximation for X at time $z_j + h$ is

$$w_{j+1} = w_j + (k_1 + 2k_2 + 2k_3 + k_4)/6. \tag{3.62}$$

3.4 Results

For an initial velocity V_{v_0} of the control beam, the value of ξ_u for the signal beam was calculated at the end of the liquid crystal cell i.e. at $z = L$. If the final value of ξ_u did not agree with the target value for the signal beam, the initial value of V_{v_0} for the control beam was then adjusted and the value of ξ_u at $z = L$ recalculated. This was done iteratively until the final position of the signal beam and the target position were sufficiently close, within 10^{-5}. To adjust the initial value of V_{v_0} the weighted least squares algorithm described in Section 2.5 was used.

The modulation theory predictions for the initial angle α_{v0} required for the signal beam to hit the target area at $z = L$ are compared with the values produced by the numerical solution of equations (2.1) –(2.3) in Figure 3.4. The initial positions of the beams are $\xi_{u0} = 13.0$ and $\xi_{v0} = -13.0$. The initial angle of the signal beam was $V_{u0} = 0$. The nonlocality was taken as $\nu = 200$ and the length of the cell was $L = 60$. It is clear that, as discussed above, the point particle approximation of Section 2.6 only yields basic agreement with the numerical results with the agreement becoming worse as the target position of the signal beam increases. As discussed in Section 3.1, the particle approximation predicts a control angle larger than the numerical value as it is based on an interaction between the beams which is too low. In contrast, the extended particle approximation of Section 3.1 gives excellent agreement with full numerical solutions over the full range of output positions. It should be noted that there is no angle α_{v0} which will route the signal beam to a position $\xi_{uf} > 13.0$. The output position of the signal beam is then not arbitrarily adjustable.

The agreement between the detailed beam trajectories as given by full numerical solutions of the nematic equations (2.1)–(2.3) and the extended modulation theory of Section 3.1 is shown in Figure 3.5. Based on the agreement shown in Figure 3.4 the trajectories of the particle approximation of Section 2.6 are not shown. Figure 3.5 shows the trajectory comparisons for both the signal u and control beams v for a range of output positions ξ_{uf} of the signal beam. As for the comparisons of Figure 3.4 the predictions of the extended particle model are in excellent agreement with the numerical solutions. The approximate trajectories for the signal beam u are identical with the numerical trajectories. This is expected as this trajectory is highly constrained as its start point ξ_{u0} and end point ξ_{uf} are fixed. In contrast, the end point ξ_{vf} of the control beam is free. Even with this freedom, the approximate trajectory for the control beam v is in excellent agreement with the numerical trajectory.

A comparison between results from the two body particle approximation with results from the extended particle model are shown in Figure 3.6. Also included are the

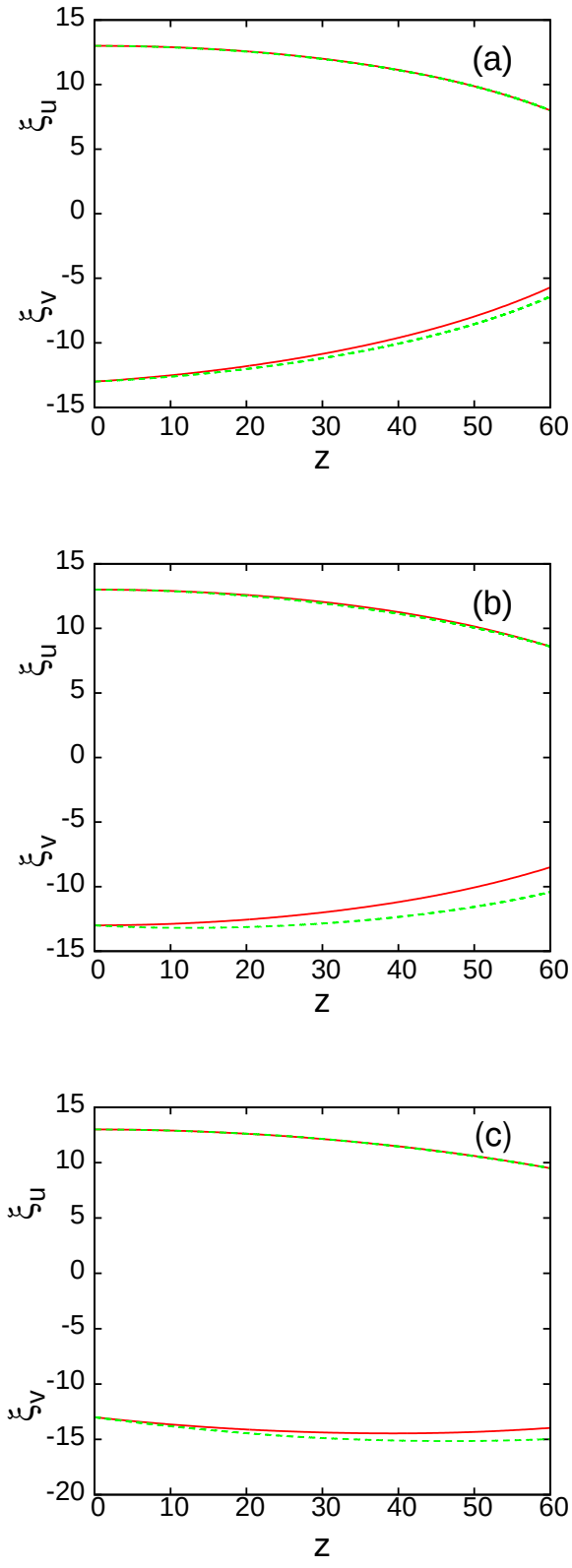

Figure 3.5: Paths of signal and control beams as given by full numerical solutions of nematic equations (2.1)–(2.3) and the extended particle. Numerical trajectory: solid (red) line, u beam upper and v beam lower; extended particle approximation: dashed (green) line, u beam upper and v beam lower. The target positions of the signal beam are (a) $\xi_{uf} = 8.0$, (b) $\xi_{uf} = 8.6$, (c) $\xi_{uf} = 9.5$. Here $\xi_{u0} = 13$, $\xi_{v0} = -13$, $\nu = 200$ and $L = 60$.

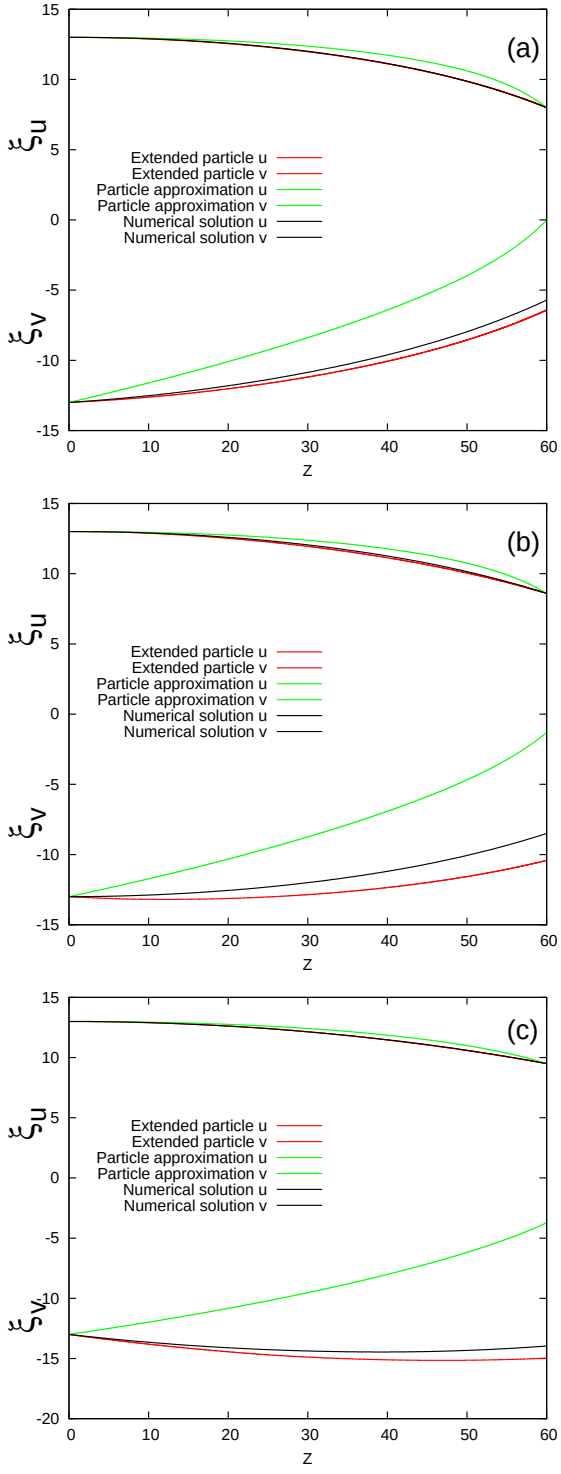

Figure 3.6: Paths of signal and control beams as given by full numerical solutions of nematic equations (2.1)–(2.3), the extended particle and the two body particle approximation. Numerical trajectory: solid (black) line, u beam upper and v beam lower; extended particle approximation: solid (red) line, u beam upper and v beam lower; two body particle approximation: solid (green) line, u beam upper and v beam lower. The target positions of the signal beam are (a) $\xi_{uf} = 8.0$, (b) $\xi_{uf} = 8.6$, (c) $\xi_{uf} = 9.5$. Here $\xi_{u0} = 13.0$, $\xi_{v0} = -13.0$,$V_{u0} = 0$, $\nu = 200$ and $L = 60$.

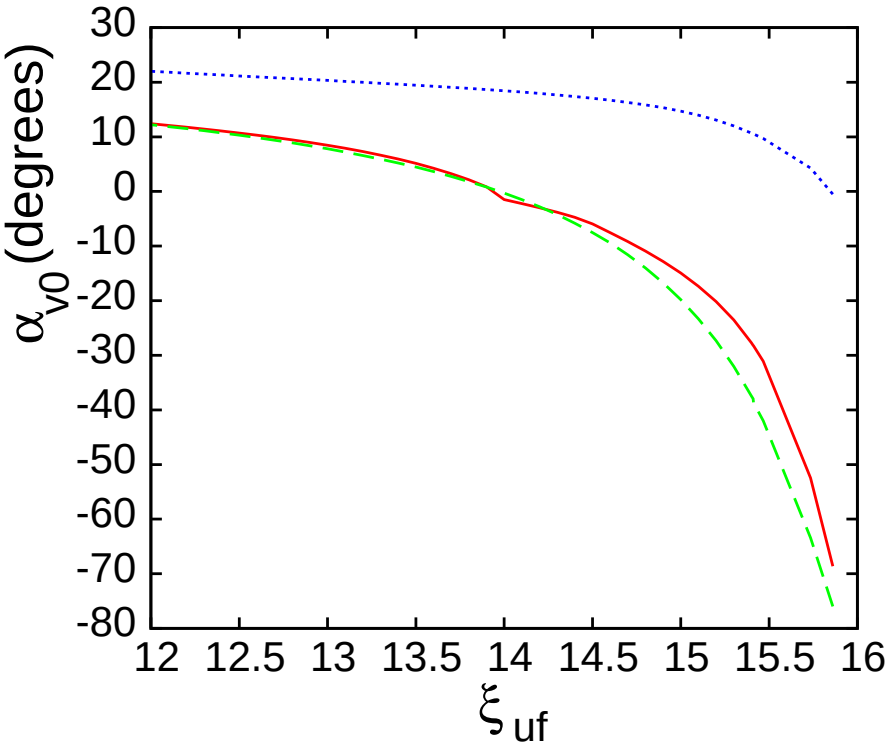

Figure 3.7: Input angle α_{v0} of control beam v needed to guide signal beam u to output position ξ_{uf}. The initial positions of the signal beam and the control beams are $\xi_{u0} = 16.0$ and $\xi_{v0} = -16.0$, respectively. Full numerical solution: dashed (green) line; particle approximation: dotted (blue) line; extended particle approximation: solid (red) line. Here $V_{u0} = 0$, $\nu = 225$ and $L = 60$.

results from the numerical solution of equations (2.1)–(2.3) for comparison.The same parameters used in Figure 3.5 were used to obtain the results shown in Figure 3.6. The trajectories predicted by the two body particle approximation show the initial angle of the control beam is greater than the angle predicted by the extended particle model and the numerical solution in each case. The attractive force predicted by the two body particle approximation is lower than the force predicted by the extended particle model, so the control beam needs to be closer to the signal beam to exert sufficient attractive force to pull the signal beam to the target position.

Figure 3.7 shows a similar comparison as for Figure 3.4 for the control beam angle α_{v0} to route the signal beam to the output position ξ_{uf}. The beams are input further apart, with $\xi_{u0} = 16$ and $\xi_{v0} = -16$ and the nonlocality is slightly higher with $\nu = 225$. The length $L = 60$ of the cell is the same. The conclusions from this new comparison are the same as those for Figure 3.4. The particle approximation of Section 2.6 only gives poor agreement with the numerical results, again due to the interaction between the beams on which the approximation is based being too low due to treating the beams as point particles, rather than extended bodies. In contrast, the extended particle approximation of Section 3.1 gives an excellent comparison with the numerical results. This is again due to it taking account of the beams being extended bodies, not point particles. The control angles α_{v0} needed to route the beam are larger than those of

Figure 3.4 as the increased separation means that the interaction between the beams is lower. As for Figure 3.4, there is a maximum deflection of the signal beam and it is not possible to find a control angle α_{v0} to route the signal beam to $\xi_{uf} > 16$, unless cross-over of the beams is permitted.

Figure 3.8 shows detailed individual beam trajectory comparisons for three of the cases of Figure 3.7. These results are very similar to the equivalent results of Figure 3.5. The agreement of the signal beam trajectory as given by the extended particle modulation theory is in perfect agreement with the numerical results. This is again due to this signal trajectory being constrained at its start and end points. This means that the trajectory between these points has little scope for variation. The extended particle modulation analysis control beam trajectories are in excellent agreement with the numerical trajectories, with some disagreement for higher values of the exit point ξ_{uf} of the signal beam. In general, as can be seen from Figures 3.5 and 3.8, the smaller the overlap between the control and signal beams, the greater the difference between the numerical and particle approximation control beam trajectories. This is because as the interaction becomes (exponentially) weaker, the more significant are small errors in the particle approximation.

Figure 3.8 can also be used to compare the results from the two body particle approximation and the extended particle model. The parameters used in these comparisons are the same as the parameters used in the comparisons shown in Figure 3.8. The initial angle of the control beam predicted by the two particle approximation is greater than the initial angle predicted by the extended particle model and as given by the numerical solution. The attractive potential force is lower in the two particle approximation than the attractive potential in the extended particle model, so to pull the signal beam to the target value, ξ_{uf}, the control beam needs to be closer to the signal beam, hence the greater initial angle. This is the same conclusion that was reached for the results shown in Figure 3.6.

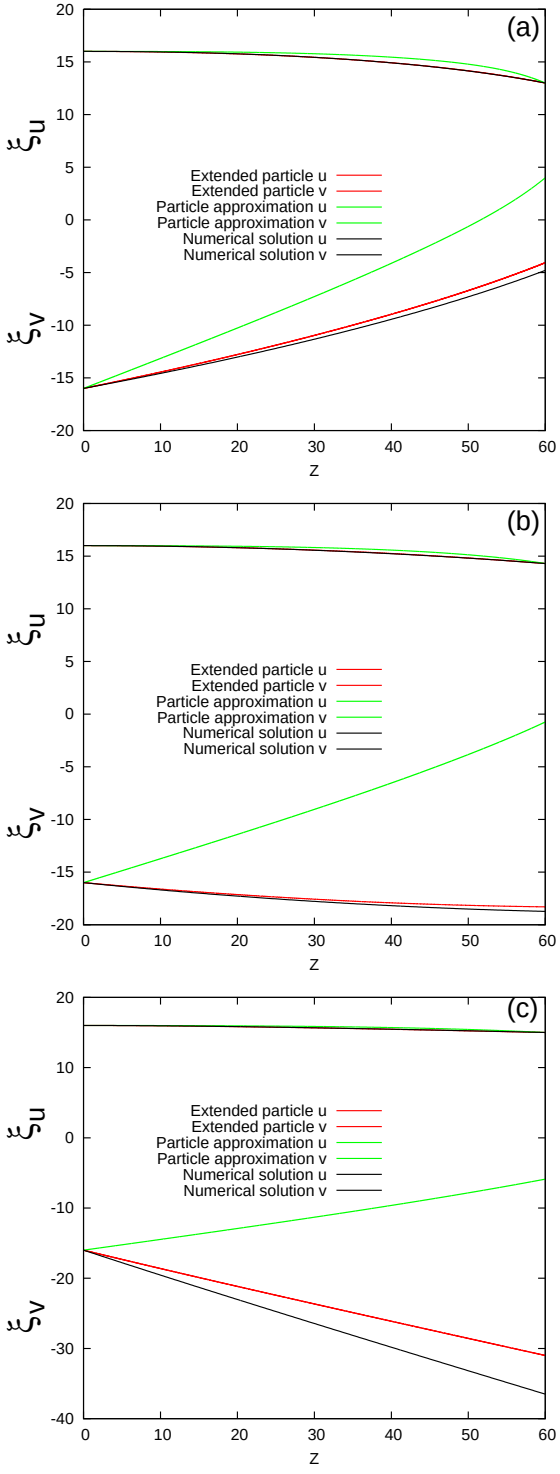

Figure 3.8: Paths of signal and control beams as given by full numerical solutions of nematic equations (2.1)–(2.3), the extended particle and the two body particle approximation. Numerical trajectory: solid (black) line, u beam upper and v beam lower; extended particle approximation: solid (red) line, u beam upper and v beam lower; two body particle approximation: solid (green) line, u beam upper and v beam lower. The target positions of the signal beam are (a) $\xi_{uf} = 13.0$, (b) $\xi_{uf} = 14.3$, (c) $\xi_{uf} = 15.0$. Here $\xi_{u0} = 16.0$, $\xi_{v0} = -16.0$, $V_{u0} = 0$, $\nu = 225$ and $L = 60$.

Chapter 4

Interaction of Dark Nematicons

The nonlinear Schrödinger (NLS) equation

$$i\frac{\partial u}{\partial z} + \frac{1}{2}\frac{\partial^2 u}{\partial z^2} \pm |u|^2 u = 0 \tag{4.1}$$

is one of the basic equations governing nonlinear wave motion.It arises as a slowly varying envelope (multiple scale) approximation in the limit of weakly nonlinear waves grouped about a central frequency, and so arises in many fields of application, including water waves and nonlinear optics [45, 55]. The NLS equation (4.1) with the plus sign on the nonlinear term is termed the focusing NLS equation and with the negative sign is termed the defocusing NLS equation [55]. The focusing NLS equation has bright soliton solutions, that is the solitons rise to a peak above the background level [45, 55, 78], while the defocusing NLS equation has dark soliton solutions, that is the solitons are a dip in a background level [55, 78]. Both forms of the NLS equation are exactly integrable via the inverse scattering transform [78].

This chapter is concerned with interacting dark solitary waves in a nematic liquid crystal doped with an azo dye [70]. Nonlinear optical beams propagating in a nematic liquid crystal are governed by an NLS-type equation for the optical beam coupled to a Poisson equation for the elastic response of the nematic [42, 44, 49, 50]. The normal optical response of the nematic results in the optical field being governed by a focusing NLS-type equation [42, 44, 49, 50]. However, adding a small concentration of an azo dye to the nematic changes the response to result in a defocusing NLS-type equation for the optical field [44, 70].

The defocusing NLS equation has an exact dark soliton solution [55]. However, the equations governing nonlinear beam propagation in a nematic liquid crystal have no exact solutions, in both the focusing and defocusing regimes [44]. For equations for which there are no exact solutions on which to base asymptotic or perturbation solutions, variational methods have been found to be useful and to give solutions in excellent agreement with both numerical solutions and experimental results [39, 73, 80, 91, 92, 93, 107, 116]. The variational method has been used to study dark solitary wave solutions of perturbed defocusing NLS equations [95, 96, 97]. The variational method has also been used to find approximations to single dark solitary wave solutions of NLS equations with nonlocal nonlinearity [98, 99]. Nonlocal equations correspond to a medium response to the optical field which depends on the whole domain and not just to the waist of the optical beam. In general, the medium response is governed by some

type of elliptic equation or is given in terms of an integral over the medium domain. The first type of response is equivalent to the second through the use of an appropriate Green's function. An example of a nonlinear, nonlocal medium is a nematic liquid crystal, for which the medium response is given by the solution of a Poisson equation for the molecular orientation [42, 44, 49, 50]. However, these examples [98, 99] of the use of the variational technique to find approximations to dark solitary wave solutions for nonlinear, nonlocal equations were for integral kernels which do not correspond to real media. The variational technique has also been employed to find dark solitary wave solutions of the equations governing a nematic liquid crystal in the defocusing regime [100, 101, 102]. It has also been employed to find approximations to the interactions between dark solitary waves [102, 103].

The success of the variational technique is based on using trial functions for the beam profiles which give a good approximation to the real profiles, as determined from numerical solutions, for example [44]. In using the variational technique to study both single and vector dark nematicons, Pu *et al* [101, 102] used a trial function for the director response which did not asymptotically account for the carrier wave(s). The appropriate use of the variational technique and the use of trial functions which have the correct asymptotic behaviour will be discussed in detail in the next section.

Previous studies of dark solitary waves and dark nematicons [97, 98, 99, 100, 101, 102, 103] have been used to find the steady dark solutions. The evolution to steady state dark solitary wave from some initial condition could not be determined from the method employed in these studies. This is because this evolution is entirely driven by the shedding of diffractive radiation [116], which was not accounted for. The variational method was extended to study the evolution of single dark solitons for the NLS equation and single dark nematicons for the defocusing nematicon equations, with good to excellent agreement being found with numerical solutions [116]. The effect of diffractive radiation was determined by linearising the equations about the background carrier wave and finding the solution of the resulting linear equation. This resulted in a loss term that was added to the variational equations, so that the solution of these equations could evolve from the initial condition to the dark steady state.

In the present work, the evolution of coupled dark nematicons for two colour optical beams [79] in a nematic liquid crystal doped with an azo dye, so that it becomes a defocusing medium [70], will be studied. Both fully dark, so that the nematicon dips to zero intensity, and grey nematicons, which dip to a finite intensity, will be studied. Perturbing the vector dark state results in the beams becoming grey and oscillating about each other. Previous approximate variational studies of steady coupled dark nematicons [101, 102] used an incorrect form for the director response, so that it was not compatible with the equations in the far field. This issue will be discussed in detail in the next section. A further study of coupled dark solitary waves in a general nonlinear, nonlocal medium [104] was again restricted to co-propagating, steady beams. Restricting the beams to be steady with the same trajectories misses the rich dynamical behaviour which results from the beams oscillating about each other in position, as found in the present work.

4.1 Governing equations

Let us consider two coherent, co-polarised light beams of different wavelength propagating through a cell filled with a nematic liquid crystal, as in the experiments of Alberucci *et al* [79]. The liquid crystal is doped with a dye so that the medium be-

comes de-focusing, as in the experiments of Piccardi *et al* [70]. Let us take the x direction to be the polarisation direction. An external bias in the x direction is applied across the cell in order to overcome the Fréedericksz threshold [64]. In the slowly varying envelope approximation the non-dimensional equations governing the propagation of the optical beams and the medium response are then [70, 79].

$$i\frac{\partial u}{\partial z} + \frac{1}{2}D_u\nabla^2 u - 2\theta u = 0, \tag{4.2}$$

$$i\frac{\partial v}{\partial z} + \frac{1}{2}D_v\nabla^2 v - 2\theta v = 0, \tag{4.3}$$

$$\nu\nabla^2\theta - 2q\theta = -2(|u|^2 + |v|^2), \tag{4.4}$$

Here u and v are slowly varying envelopes of the electric fields of the light beams and θ is the extra rotation of the nematic molecules from their pre-tilt state due to the optical beams. The parameter ν measures strength of the response of the nematic medium to the optical beams and is large, $O(100)$, in experimental cells [80]. The parameter q is proportional to the square of the external bias field [49, 50, 80]. It should be noted that in the electric field equations (4.2) and (4.3) the coefficants of $2\theta u$ and $2\theta v$ have been set equal, and normalised to 1. These coefficients are not equal [79]. However, these coefficients differ by only a few percent [79], and so have been set equal for simplicity. A similar comment applies to the coefficients of $2|u|^2$ and $2|v|^2$ in the director equation (4.4). The two colour dark nematicons form on linear carrier waves. These carrier waves are

$$u = Ue^{-2i(U^2+V^2)z/q} \qquad \text{and} \qquad v = Ve^{-2i(U^2+V^2)z/q} \tag{4.5}$$

as these are the linear wave solutions of the nematic equations (4.2) – (4.4). While the governing equations (4.2) – (4.4) have been introduced in the context of nonlinear optical beam propagation in nematic liquid crystals, the equations themselves are, in fact, general and also govern nonlinear optical beam propagation in thermo-optic media [83], such as lead glasses [84, 85, 86], and certain photorefractive crystals [87]. In general, equations similiar to (4.2) – (4.4) arise when the nonlinear response of the medium is accompanied by some sort of diffusive mechanism [82]. In addition, similar equations arise in models of turbulence in fluids [88, 89].

The two colour dark nematicon equations (4.2) – (4.4) have the Lagrangian formulation

$$\begin{aligned} L = {} & i(u^*u_z - uu_z^*) - D_u|u_x|^2 - 4\theta|u|^2 + i(v^*v_z - vv_z^*) \\ & - D_v|v_x|^2 - 4\theta|v|^2 + \nu\theta_x^2 + 2q\theta^2 - \frac{2(U^2+V^2)^2}{q}. \end{aligned} \tag{4.6}$$

The two colour dark nematicon equations (4.2) – (4.4) have no known exact solutions. In the absence of such known solutions, variational approximations have been found to give results in excellent agreement with numerical solutions and experimental results [39, 73, 80, 91, 92, 116]. In previous work [116] a suitable profile for a dark nematicon was found to be based on the dark soliton solution of the de-focusing NLS equation

[55]. Extending this to two colour nematicons results in the trial functions

$$u = \left[B_u \tanh\frac{x-\xi_u}{w_u} + iA_u\right] e^{-2i(U^2+V^2)z/q}, \quad B_u^2 + A_u^2 = U^2, \tag{4.7}$$

$$v = \left[B_v \tanh\frac{x-\xi_v}{w_v} + iA_v\right] e^{-2i(U^2+V^2)z/q}, \quad B_v^2 + A_v^2 = V^2, \tag{4.8}$$

$$\theta = \frac{1}{q}\left[\alpha_u^2 \tanh^2\frac{x-\xi_u}{\beta_u} + \alpha_v^2 \tanh^2\frac{x-\xi_v}{\beta_v} + \gamma^2\right], \quad \alpha_u^2 + \alpha_v^2 + \gamma^2 = U^2 + V^2. \tag{4.9}$$

It is noted that the nematicons are fully dark if $A_u = 0$ and $A_v = 0$. For $A_u \neq 0$ and $A_v \neq 0$ the nematicons are strictly termed grey. In previous work [101, 102] it was assumed that the medium response was given by

$$\theta = \alpha\left[\tanh^2\frac{x-\xi}{w} - 1\right] \tag{4.10}$$

for a single dark nematicon and

$$\theta = \left[\alpha_u \operatorname{sech}^2\frac{x-\xi_u}{w_u} - 1\right]\left[\alpha_v \operatorname{sech}^2\frac{x-\xi_v}{w_v} - 1\right] - 1 \tag{4.11}$$

for two coupled dark nematicons. We note that both these trial functions for the director go to 0 as $x \to \pm\infty$. However, we note that the optical fields u and v must approach the carrier waves (4.5) as $x \to \pm\infty$ [55], in agreement with the optical field trial functions used in Pu *et al* [101, 102]. The director equation (4.4) then shows that the director approaches a constant as $x \to \pm\infty$, the same constant as in the trial function (4.9). The director trial functions used in Pu *et al* [101, 102] are therefore inconsistent with the governing equations.

Substituting the trial functions (4.7)–(4.9) into the Lagrangian (4.6) and averaging in x [45], that is integrating in x from $-\infty$ to ∞, results in the average Lagrangian

$$\begin{aligned}
\mathcal{L} = &-4\left(B_uA_u - U^2\tan^{-1}\frac{B_u}{A_u}\right)\xi_u' - \frac{4}{3}\frac{B_u^2}{w_u} \\
&- \frac{4\sqrt{\pi}C_1C_2\beta_u w_u\alpha_u^2B_u^2}{q\sqrt{C_1^2\beta_u^2 + C_2^2w_u^2}} - \frac{4\sqrt{\pi}C_1C_2\beta_v w_u\alpha_v^2B_u^2}{q\sqrt{C_1^2\beta_v^2 + C_2^2w_u^2}}e^{-(\xi_u-\xi_v)^2/(C_1^2\beta_v^2+C_2^2w_u^2)} \\
&- 4\left(B_vA_v - V^2\tan^{-1}\frac{B_v}{A_v}\right)\xi_v' - \frac{4}{3}\frac{B_v^2}{w_v} \\
&- \frac{4\sqrt{\pi}C_1C_2\beta_v w_v\alpha_v^2B_v^2}{q\sqrt{C_1^2\beta_v^2 + C_2^2w_v^2}} - \frac{4\sqrt{\pi}C_1C_2\beta_u w_v\alpha_u^2B_v^2}{q\sqrt{C_1^2\beta_u^2 + C_2^2w_v^2}}e^{-(\xi_u-\xi_v)^2/(C_1^2\beta_u^2+C_2^2w_v^2)} \\
&+ \frac{16\nu}{15q^2}\left[\frac{\alpha_u^4}{\beta_u} + \frac{\alpha_v^4}{\beta_v}\right] + \frac{8}{3q}[\alpha_u^4\beta_u + \alpha_v^4\beta_v] - \frac{4\sqrt{\pi}C_3\beta_u\beta_v\alpha_u^2\alpha_v^2}{q\sqrt{\beta_u^2+\beta_v^2}}e^{-(\xi_u-\xi_v)^2/[C_3^2(\beta_u^2+\beta_v^2)]} \\
&+ \frac{8\nu\sqrt{\pi}}{C_3q^2}\frac{\alpha_u^2\alpha_v^2\beta_u\beta_v}{(\beta_u^2+\beta_v^2)^{3/2}}\left[\frac{1}{2} - \frac{(\xi_u-\xi_v)^2}{C_3^2(\beta_u^2+\beta_v^2)}\right]e^{-(\xi_u-\xi_v)^2/[C_3^2(\beta_u^2+\beta_v^2)]}.
\end{aligned} \tag{4.12}$$

The constants C_1, C_2 and C_3 occurring in this averaged Lagrangian are

$$C_1 = \frac{2\sqrt{6}}{\pi^{3/2}}, \quad C_2 = \frac{2}{\sqrt{\pi}}, \quad C_3 = \frac{4\sqrt{2}}{3\sqrt{\pi}}. \tag{4.13}$$

The inverse tan terms have been added to the averaged Lagrangian (4.12) in order to subtract out the momentum of the background carrier waves [95, 96].

Taking variations of the averaged Lagrangian (4.12) with respect to the nematicon parameters B_u, w_u, ξ_u, α_u, β_u results in the variational, or modulation [45], equations governing the evolution of the two colour dark nematicons as

$$\delta B_u: \quad \frac{B_u}{A_u}\frac{d\xi_u}{dz} = \frac{1}{3w_u} + \frac{\sqrt{\pi}C_1C_2\beta_u w_u\alpha_u^2}{q\sqrt{C_1^2\beta_u^2 + C_2^2w_u^2}} + \frac{\sqrt{\pi}C_1C_2\beta_v w_u\alpha_v^2}{q\sqrt{C_1^2\beta_v^2 + C_2^2w_u^2}}e^{-(\xi_u-\xi_v)^2/(C_1^2\beta_v^2+C_2^2w_u^2)}, \tag{4.14}$$

$$\begin{aligned}\delta\xi_u: \quad \frac{B_u^2}{A_u}\frac{dB_u}{dz} &= \frac{\sqrt{\pi}C_1C_2\beta_v w_u\alpha_v^2B_u^2(\xi_u-\xi_v)}{qC_1^2\beta_v^2 + C_2^2w_u^{2\,3/2}}e^{-(\xi_u-\xi_v)^2/(C_1^2\beta_v^2+C_2^2w_u^2)} \\ &+ \frac{\sqrt{\pi}C_1C_2\beta_u w_v\alpha_u^2B_v^2(\xi_u-\xi_v)}{qC_1^2\beta_u^2 + C_2^2w_v^{2\,3/2}}e^{-(\xi_u-\xi_v)^2/(C_1^2\beta_u^2+C_2^2w_v^2)} \\ &+ \frac{\sqrt{\pi}\beta_u\beta_v\alpha_u^2\alpha_v^2(\xi_u-\xi_v)}{qC_3^2(\beta_u^2+\beta_v^2)^{3/2}}e^{-(\xi_u-\xi_v)^2/(C_3^2(\beta_u^2+\beta_v^2))} \\ &+ \frac{2\nu\sqrt{\pi}\alpha_u^2\alpha_v^2(\xi_u-\xi_v)}{q^2C_3^2(\beta_u^2+\beta_v^2)}\left[\frac{3\beta_u\beta_v}{2(\beta_u^2+\beta_v^2)^{3/2}} - \frac{\beta_u\beta_v(\xi_u-\xi_v)^2}{C_3^2(\beta_u^2+\beta_v^2)^{5/2}}\right] \\ &\times e^{-(\xi_u-\xi_v)^2/(C_3^2(\beta_u^2+\beta_v^2))},\end{aligned} \tag{4.15}$$

$$\begin{aligned}\delta w_u: \quad \frac{D_uB_u^2}{3w_u^2} &= \frac{\sqrt{\pi}C_1^3C_2\beta_u^3\alpha_u^2B_u^2}{q(C_1^2\beta_u^2 + C_2^2w_u^2)^{3/2}} \\ &+ \left[\frac{\sqrt{\pi}C_1^3C_2\beta_v^3\alpha_v^2B_u^2}{q(C_1^2\beta_v^2 + C_2^2w_u^2)^{3/2}} + \frac{2\sqrt{\pi}C_1C_2^3\beta_v w_u^2\alpha_v^2B_u^2(\xi_u-\xi_v)^2}{qC_1^2\beta_v^2 + C_2^2w_u^{2\,5/2}}\right] \\ &\times e^{-(\xi_u-\xi_v)^2/(C_1^2\beta_v^2+C_2^2w_u^2)},\end{aligned} \tag{4.16}$$

$$\begin{aligned}\delta\alpha_u: \quad \frac{4}{3}\left(\frac{2\nu}{5q\beta_u^2}+1\right)\alpha_u^2 &= \frac{\sqrt{\pi}C_1C_2w_uB_u^2}{\sqrt{C_1^2\beta_u^2 + C_2^2w_u^2}} \\ &+ \frac{\sqrt{\pi}C_1C_2w_v\beta_v^2}{\sqrt{C_1^2\beta_u^2 + C_2^2w_u^2}}e^{-(\xi_u-\xi_v)^2/(C_1^2\beta_u^2+C_2^2w_v^2)} - \frac{\sqrt{\pi}C_3\beta_v\alpha_v^2}{\sqrt{\beta_u^2+\beta_v^2}} \\ &\times e^{-(\xi_u-\xi_v)^2/(C_3^2(\beta_u^2+\beta_v^2))} \\ &- \frac{2\sqrt{\pi}\nu\alpha_v^2B_u^2}{qC_3}\left[\frac{\beta_v}{2(\beta_u^2+\beta_v^2)^{3/2}} - \frac{(\xi_u-\xi_v)^2\beta_v}{C_3^2(\beta_u^2+\beta_v^2)^{5/2}}\right]e^{-(\xi_u-\xi_v)^2/(C_3^2(\beta_u^2+\beta_v^2))},\end{aligned} \tag{4.17}$$

$$
\begin{aligned}
\delta B_u: \quad & \frac{2}{3}\left(1-\frac{2\nu}{5q\beta_u^2}\right)\alpha_u^2 = \frac{\sqrt{\pi}C_1C_2^3w_u^3B_u^2}{(C_1^2\beta_u^2+C_2^2w_u^2)^{3/2}} \\
& + \frac{\sqrt{\pi}C_1C_2^3w_v^3B_v^2}{(C_1^2\beta_u^2+C_2^2w_v^2)^{3/2}}e^{-(\xi_u-\xi_v)^2/(C_1^2\beta_u^2+C_2^2w_v^2)} \\
& + \frac{2\sqrt{\pi}C_1^3C_2\beta_u^2w_vB_v^2(\xi_u-\xi_v)^2}{(C_1^2\beta_u^2+C_2^2w_v^2)^{5/2}}e^{-(\xi_u-\xi_v)^2/(C_1^2\beta_u^2+C_2^2w_v^2)} \\
& - \frac{\sqrt{\pi}C_3\beta_v^3\alpha_v^2}{(\beta_u^2+\beta_v^2)^{3/2}}e^{-(\xi_u-\xi_v)^2/(C_3^2(\beta_u^2+\beta_v^2))} - \frac{2\sqrt{\pi}\beta_u^2\beta_v\alpha_v^2(\xi_u-\xi_v)^2}{C_3(\beta_u^2+\beta_v^2)^{5/2}}e^{-(\xi_u-\xi_v)^2/(C_3^2(\beta_u^2+\beta_v^2))} \\
& - \frac{2\sqrt{\pi}\nu\alpha_v^2\beta_u^2}{qC_3}\left[\frac{\beta_v^2-2\beta_u^2}{2(\beta_u^2+\beta_v^2)^{5/2}} - \frac{(\beta_v^2-5\beta_u^2)(\xi_u-\xi_v)^2}{C_3^2(\beta_u^2+\beta_v^2)^{7/2}} - \frac{2(\xi_u-\xi_v)^4\beta_u^2}{C_3^4(\beta_u^2+\beta_v^2)^{9/2}}\right] \\
& \times e^{-(\xi_u-\xi_v)^2/(C_3^2(\beta_u^2+\beta_v^2))}. \qquad (4.18)
\end{aligned}
$$

There are also 5 symmetric equations obtained by taking variations with respect to B_v, w_v, ξ_v, α_v and β_v. In taking these variations, it should be noted that A_u, B_u and A_v, B_v are not independent parameters as $A_u^2+B_u^2=U^2$ and $A_v^2+B_v^2=V^2$. These modulation equations are more complicated than those for a single dark nematicon [116] due to the exponential coupling terms between the two beams. Indeed, the modulation equations for a single dark nematicon were found to have an exact solution [116]. The modulation equations (4.14)–(4.18) will be solved numerically using the standard fourth order Runge-Kutta scheme.

The modulation equations (4.14)–(4.18) govern both totally dark and grey two colour nematicons. Two colour dark nematicons have $A_u=A_v=0$. The modulation equation (4.15) and its v counterpart show that the nematicons then have fixed depth and remain dark, as B_u and B_v do not evolve. Furthermore, the modulation equation (4.14) and its v counterpart show that the dark nematicons are fixed in position. These two colour dark nematicons can have any separation $|\xi_u-\xi_v|$. The modulation equations also show that this steady state of two colour dark nematicons is unstable when the diffusion coefficients are not equal. Physically the beam with the higher diffusion coefficient radiates more energy and decays as the the beam evolves. The following analysis shows mathematically why this is so.

It is found from numerical solutions that the coupled black nematicons are unstable when the diffraction coefficients are different, $D_u \neq D_v$. This instability is illustrated in Figures 4.4, 4.5 and 4.6. It can be seen from Figure 4.4 that for black beams, the v beam, the beam with the higher diffraction coefficient, spreads and decays into radiation, while the u beam evolves to a single black nematicon. The fully dark nematicons shown in Figure 4.6 show this instability. The v beam decays as it evolves, while the u beam shows a gain in height over the same evolution. This instability also arises for coupled black solitary waves governed by coupled defocusing NLS equations with different diffraction coefficients [75]

$$
\begin{aligned}
i\frac{\partial u}{\partial z}+\frac{D_u}{2}\frac{\partial^2 u}{\partial x^2}-\left(|u|^2+|v|^2\right)u &= 0, \\
i\frac{\partial v}{\partial z}+\frac{D_v}{2}\frac{\partial^2 v}{\partial x^2}-\left(|u|^2+|v|^2\right)v &= 0. \qquad (4.19)
\end{aligned}
$$

Again, the beam with the higher diffraction coefficient decays into radiation and the other beam evolves to a single black soliton, as in Figure 4.6.

This instability of coupled black nematicons can be understood on physical grounds

as follows. The higher diffraction coefficient of the (v) beam causes it to diffract more than the u beam. As the u beam is coupled to the v beam, the widening of the v beam causes the u beam to deform, reinforcing the widening of the v beam. The widening of the v beam is accompanied by the shedding of diffractive radiation, which causes it to decay. A non-standard behaviour of this decay of the v beam is that the shed radiation grows in amplitude. This can be understood by noting that the u and v beams individually conserve power, or mass in terms of invariances of the Lagrangian for the coupled equations [78]. These optical powers are given by

$$P_u = \int_{-\infty}^{\infty} \left(|U|^2 - |u|^2\right)\, dx, \tag{4.20}$$

$$P_v = \int_{-\infty}^{\infty} \left(|V|^2 - |v|^2\right)\, dx. \tag{4.21}$$

Hence, for the v beam to decay, this decay must be accompanied by radiation rising above $v = V$ to balance the decay to 0 in an expanding vicinity of the nematicon minimum at $x = 0$. Thus the instability of the v beam occurs by the shedding of diffractive radiation of growing amplitude.

This physical instability mechanism can be further analysed through the following mathematical argument. The coupled dark nematicon equations are (4.2)–(4.4). Taking the diffraction coefficients as being nearly equal, so that $D_v = D_u + \epsilon$, $|\epsilon| \ll 1$. This is the experimental case as the diffraction coefficients for red and infra-red light differ by about 5% [80]. We then expand

$$u = u_0 + \epsilon u_1 + \ldots, \quad v = v_0 + \epsilon v_1 + \ldots, \quad \theta = \theta_0 + \epsilon\theta_1 + \ldots \tag{4.22}$$

At $O(1)$, $u_0 = v_0$, where

$$i\frac{\partial u_0}{\partial z} + \frac{D_u}{2}\frac{\partial^2 u_0}{\partial x^2} - 2\theta_0 u_0 = 0, \tag{4.23}$$

$$\nu\frac{\partial^2 \theta_0}{\partial x^2} - 2q\theta_0 = -4|u_0|^2. \tag{4.24}$$

At $O(\epsilon)$ we have

$$i\frac{\partial u_1}{\partial z} + \frac{D_u}{2}\frac{\partial^2 u_1}{\partial x^2} - 2\theta_0 u_1 - 2\theta_1 u_0 = 0, \tag{4.25}$$

$$i\frac{\partial v_1}{\partial z} + \frac{D_u}{2}\frac{\partial^2 v_1}{\partial x^2} - 2\theta_0 v_1 - 2\theta_1 v_0 = -\frac{1}{2}\frac{\partial^2 v_0}{\partial x^2} = -\frac{1}{2}\frac{\partial^2 u_0}{\partial x^2}. \tag{4.26}$$

The equation for θ_1 is not needed. Let us set $Z = u_1 - v_1$ and subtract (4.25) and (4.26), to give

$$i\frac{\partial Z}{\partial z} + \frac{D_u}{2}\frac{\partial^2 Z}{\partial x^2} - 2\theta_0 Z = \frac{1}{2}\frac{\partial^2 u_0}{\partial x^2}. \tag{4.27}$$

We now look for soliton solutions for u_0 and Z

$$u_0 = U_0(x)e^{i\sigma z}, \quad Z = f(x)e^{i\sigma z}. \tag{4.28}$$

From (4.27) we have

$$\frac{D_u}{2} f'' - (\sigma + 2\theta_0)\, f = \frac{1}{2}\frac{d^2 U_0}{dx^2}. \tag{4.29}$$

From equation (4.23) for u_0 we have

$$\frac{D_u}{2}\frac{d^2 U_0}{dx^2} - (\sigma + 2\theta_0)\, U_0 = 0. \tag{4.30}$$

This equation becomes

$$\frac{d^2 U_0}{dx^2} = \frac{2}{D_u}(\sigma + 2\theta_0)\, U_0. \tag{4.31}$$

Using this, equation (4.29) for f becomes

$$\frac{D_u}{2} f'' - (\sigma + 2\theta_0)\, f = \frac{\sigma}{D_u} U_0 + \frac{2}{D_u}\theta_0 U_0. \tag{4.32}$$

We now note that U_0 is a solution of the homogeneous equation, so that the forcing component U_0/D_u is resonant. There is then no coupled black solitary wave solution for the coupled dark nematicon equations for all ν.

Note that from the modulation equations (4.14)–(4.18) perturbing A_u and A_v away from 0 causes B_u, B_v and ξ_u, ξ_v to evolve. Numerical solutions of the next section show that the nematicons oscillate about each other in position with their depths evolving in a periodic fashion.

4.2 Solving the dark nematicon equations

The coupled dark nematicon equations (4.2)–(4.4) are solved using an extension of the techniques described by Fornberg and Whitham in their 1978 paper [115]. The initial profiles of the optical beams are given by equation (4.7) and (4.8). This numerical technique is described in detail in Section 2.3.

The modulation equations (4.14)–(4.18) are solved using the same techniques described in Section 3.3. The algebraic equations (4.16)–(4.18) are first solved to find the variables α_u, β_u, w_u, α_v, β_v and w_v. This is accomplished by using the method of steepest descent to find an initial approximation for these variables and then passing the approximate values to a Newton-Rhapson technique [117] to obtain values with greater accuracy. These variables are then used to solve the differential equations portion of the modulation equations (4.14)–(4.15). The equations (4.14)–(4.15) and their v equivalents form a 4×4 system of equations that is solved using a fourth order Runge-Kutta method that is described in Section 2.7. Solving this system of differential equations gives the variables B_u, B_v, ξ_u and ξ_v.

4.3 Results

In this section the numerical solutions of equations (4.2)–(4.4) are compared with the solutions of the modulation equations (4.14)–(4.18). As mentioned in the previous section, equations (4.2)–(4.4) were solved using an extension of techniques used by Fornberg and Whitham [115]. Step sizes of $\Delta x = 0.1$ and $\Delta z = 0.002$ were used with a spatial interval of 1024 in the x direction. The modulation equations were solved with

a fixed step size of 0.005 for the differential equations. The value of ν was set to 200 generally. This is a value that is consistent with experimental values of ν [50, 63]. When studying instabilities of nematicons when the diffraction coefficients of the two beams are unequal, for some cases the value of ν was set to 0.1 to represent the limiting case of ν approaching 0. When nematicons with equal diffraction coefficients were studied the values of D_u and D_v were set to 1.00, when unequal diffraction coefficients were studied, D_u and D_v were set to 1.0 and 1.25 respectively. The amplitudes of the optical beams are $U = 0.5$ and $V = 0.5$, with the initial values of A_u and A_v being 0.2. The corresponding values of B_u and B_v are then 0.458 as the values of A, B and U, V are linked due to equations (4.7) and (4.8). The values of ξ_{u0} and ξ_{v0} are 1.0 and -1.0 respectively.

Figure 4.1 shows a comparison between the full numerical and modulation solutions for depths of the beams, which are $|u|$ and $|v|$ at their minimum. The depth is measured by the parameters A_u and A_v. The solution u is of the form $u = [B_u \tanh \frac{x-\xi_u}{w_u} + iA_u]e^{-kz}$, so $|u|$ has a minimum value of $|A_u|$ when $x = \xi_u$, similarly $|v|$ has a minimum value of $|A_v|$. Figure 4.1 shows that the period of the full numerical solution is a little shorter than the period of the modulation solution and the depth of modulation solution is damped in relation to the full numerical solution. The full numerical solution and the modulation solutions show good agreement between the mean depth of both optical beams. The modulation equations form a nonlinear oscillator with the depth and the positions linked. The lower depth of the modulation solutions results in the depth period being longer than the period for the full numerical solution [73].

Figures 4.2 and 4.3 show comparisons between the full numerical solutions and the modulation solutions for the positions of the optical beams. The beams oscillate around each other. Equation (4.15) shows that if ξ_u is greater than ξ_v, then $\frac{dB_u}{dz}$ is positive, so B_u is increasing and A_u will decrease because of equation (4.7). From equation (4.14) $\frac{d\xi_u}{dz}$ is always positive. However with B_u increasing and A_u decreasing, $\frac{d\xi_u}{dz}$ will decrease and the beam will slow down. The opposite is true for the v optical beam, so the v beam will speed up. There is a slight period difference between the numerical and the approximate periods of the oscillation. The net result is that the two beams will oscillate about each other in terms of position and amplitude. Figure 4.2 (a) shows the comparison for beam u between the full numerical solution and the modulation solution and Figure 4.2 (b) shows the same comparison for the v beam. Subjectively the full numerical and modulation solutions show similar graphs, however the is a difference in the phase and amplitude of the solutions showing that the modulation equations are not capturing all the features of the full numerical solution. Figure 4.3 (a) shows the position of u and v beams as given by the full numerical solutions, while Figure 4.3 (b) shows the same results for the u and v beams as given by the modulation equations. There is excellent agreement between these two graphs.

A result found from the analysis of the couple dark nematicon equations (4.2)–(4.4) is that the dark nematicons are unstable when the diffraction coefficients D_u and D_v are not equal. Figure 4.4 shows the results when $D_u = 1.00$, $D_v = 1.25$ and $\nu = 0.01$. The value for ν was chosen as the coupled dark nematicon equations (4.2)–(4.4) approach the coupled NLS equations in the limit of ν approaching 0. Figure 4.4 shows the optical beams in profile. Part (a) of the figure is up to distance $z = 147$ and part (b) is the profile at distance $z = 150$.

Figure 4.6 shows the evolution of black u and v beams when the values of the diffraction coefficients are different. Both the u and v beams are fully dark beams (the depth of both beams reaches 0 and there is no evolution of the position) with

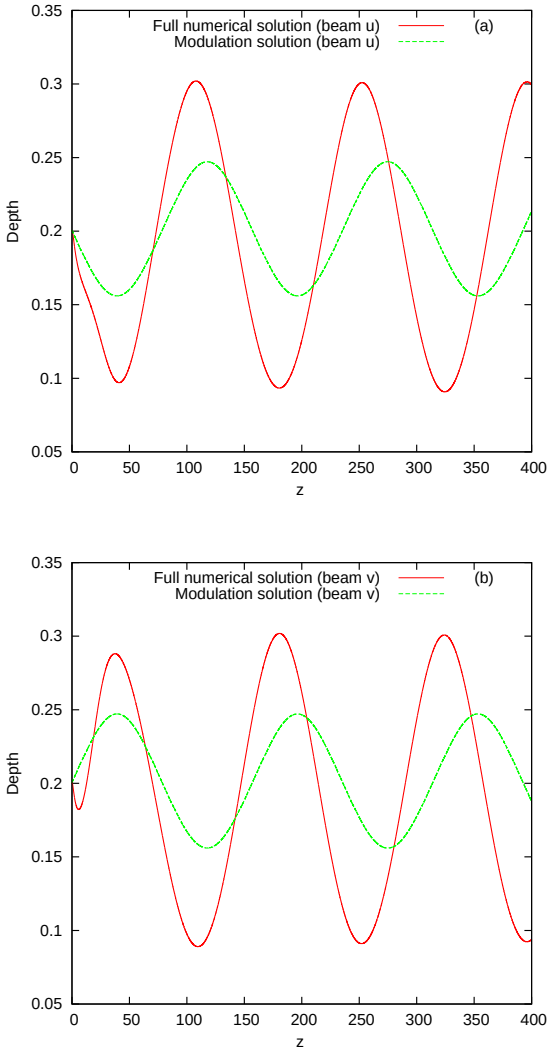

Figure 4.1: Depths of dark nematicons from full numerical solution of equations (4.2)–(4.4) compared with solutions of modulation equations. (a) u beam, full numerical solution: solid (red) line; modulation equations: dashed (green) line. (b) v beam, full numerical solution: solid (red) line; modulation equations: dashed (green) line. Parameters are $\xi_{u0} = 1.0$, $\xi_{v0} = -1.0$, $\nu = 200$ and $L = 400$.

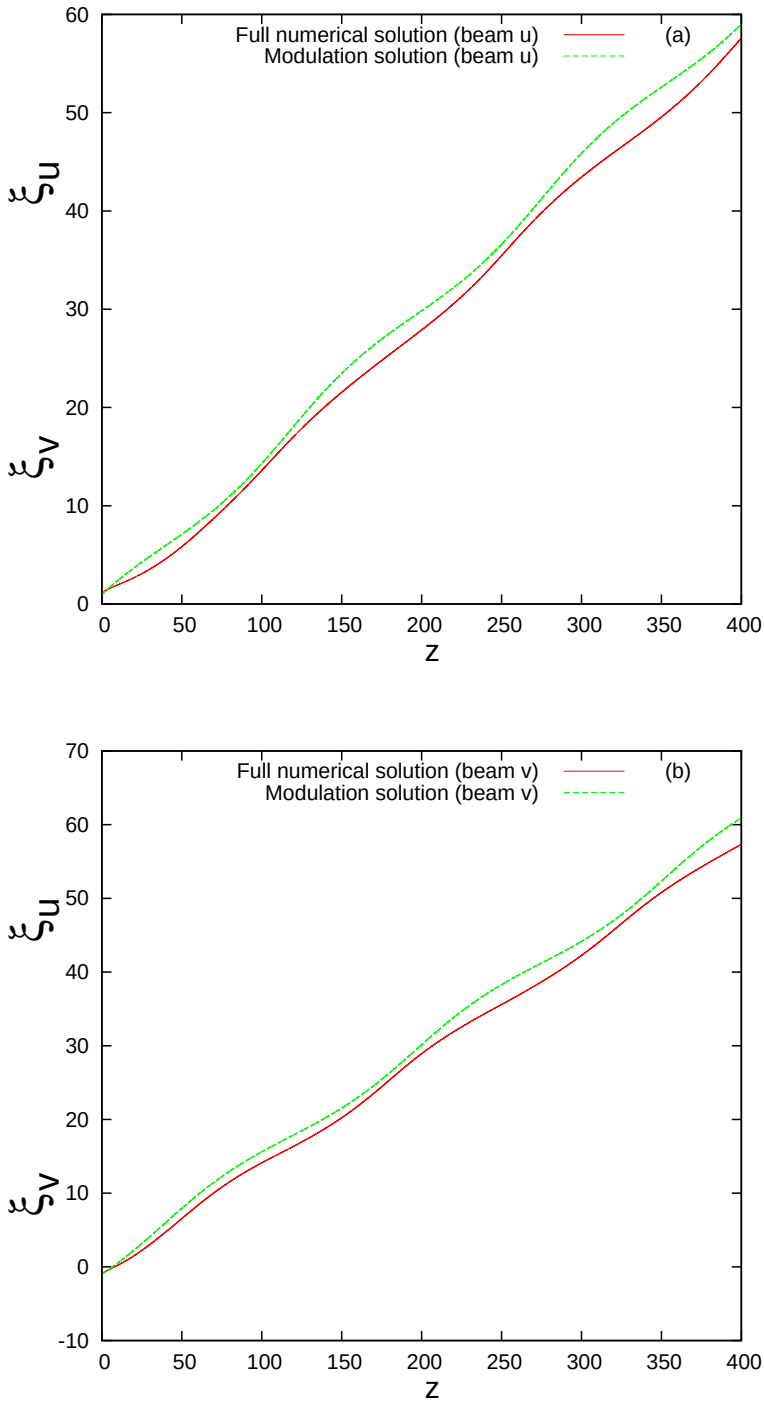

Figure 4.2: The position of dark nematicons from full numerical solution of equations (4.2)–(4.4) compared with solutions of modulation equations. (a) u beam, full numerical solution: solid (red) line; modulation equations: dashed (green) line. (b) v beam, full numerical solution: solid (red) line; modulation equations: dashed (green) line. Parameters are $\xi_{u0} = 1.0$, $\xi_{v0} = -1.0$, $\nu = 200$ and $L = 400$.

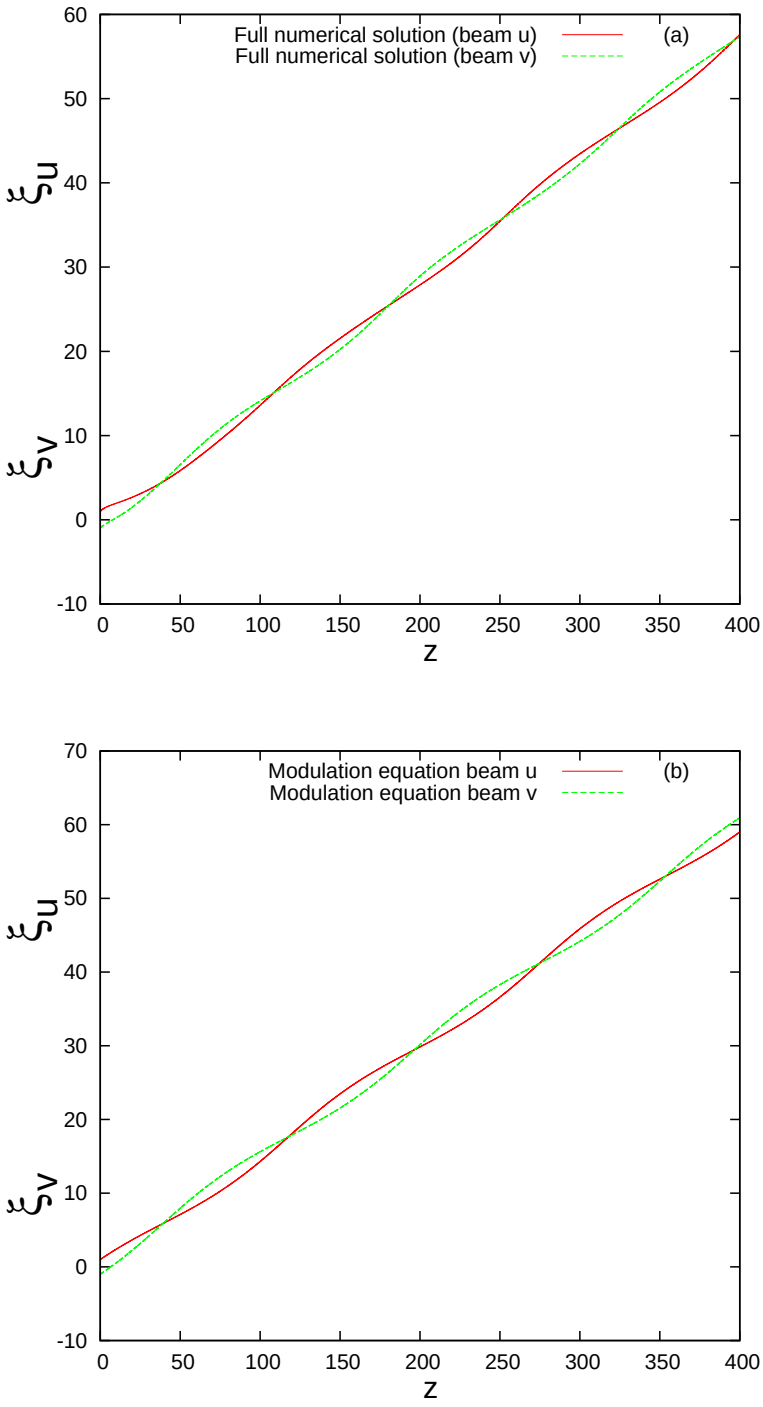

Figure 4.3: The positions of dark nematicons from full numerical solution of equations (4.2) – (4.4) compared with solutions of modulation equations. (a) full numerical solution: u beam solid (red) line; v beam dashed (green) line. (b) modulation equations: u beam solid (red) line; v beam dashed (green) line. Parameters are $\xi_{u0} = 1.0$, $\xi_{v0} = -1.0$, $\nu = 200$ and $L = 400$.

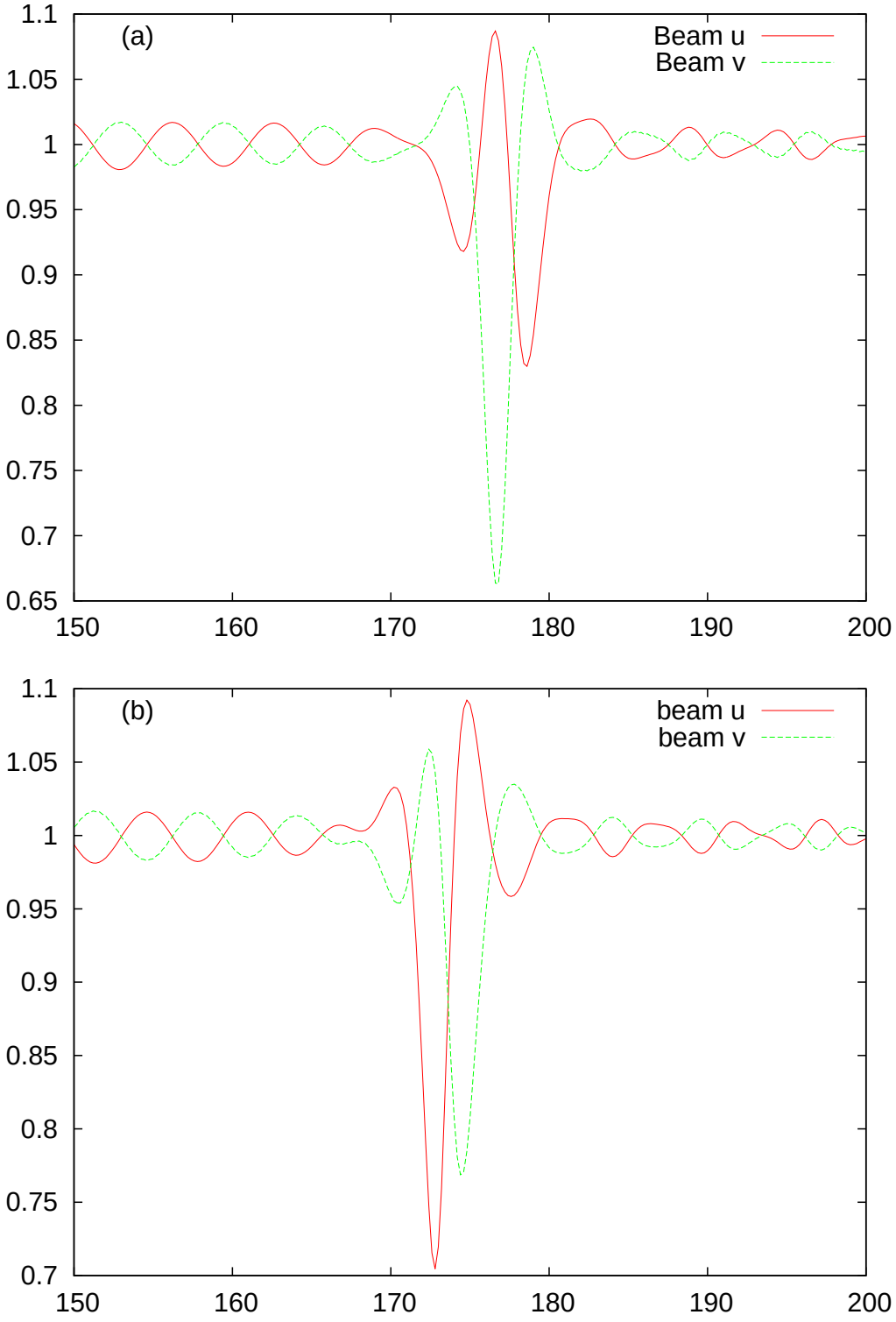

Figure 4.4: Profile of full numerical solution in the x direction. The solutions are from equations (4.2)–(4.4) with unequal diffraction coefficients. u beam solid (red) line; v beam dashed (green) line. (a) Time of evolution 147 (b) Time of evolution 150. Parameters are $\xi_{u0} = 1.0$, $\xi_{v0} = -1.0$, $\nu = 0.1$, $D_u = 1.00$, $D_v = 1.25$.

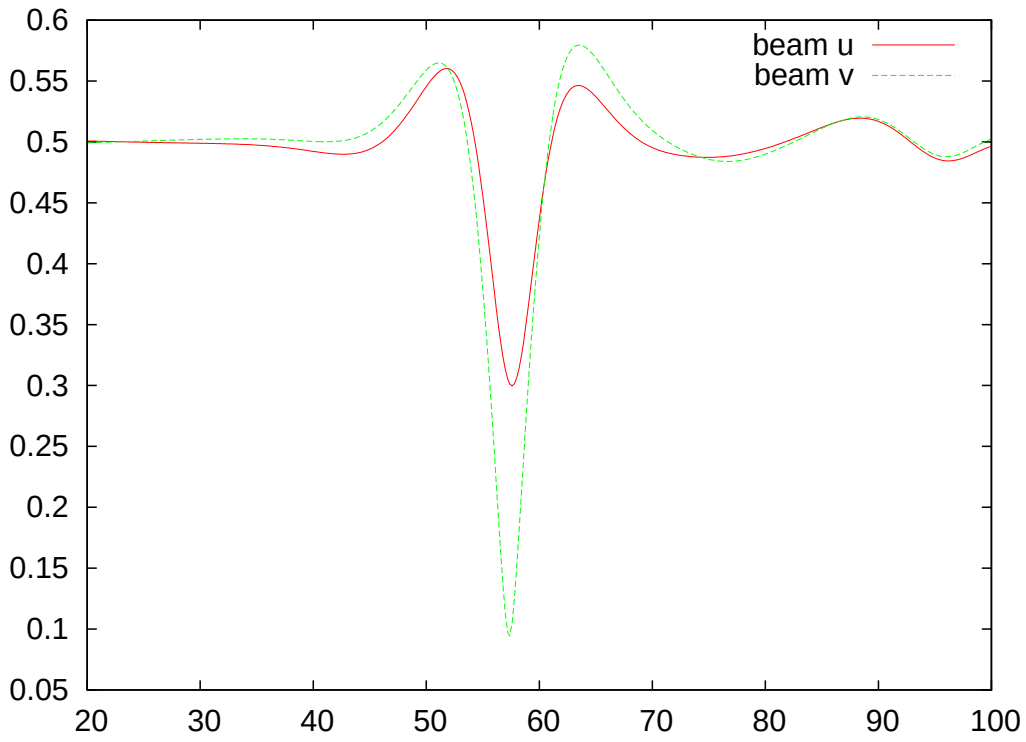

Figure 4.5: Profile of full numerical solution in the x direction. The solutions are from equations (4.2)–(4.4) with unequal diffraction coefficients. u beam solid (red) line; v beam dashed (green) line. Time of evolution is 150. Parameters are $\xi_{u0} = 1.0$, $\xi_{v0} = -1.0$, $\nu = 200$, $D_u = 1.00$, $D_v = 1.25$.

$A_u = A_v = 0$. The value of ν is 0.1. The value of D_u is 1.00 and the value of D_v is 1.25. Figure 4.6 shows the v beam is becoming increasingly unstable as it evolves. The radiation loss is causing the v beam to decay by expanding in width, as explained above. On the other hand, the amplitude of the u beam increases to a value of 1.40 at evolution distance $z = 200$. The u beam is stable, while the v beam eventually dies [75].

The modulation equations (4.14)–(4.18) do not show any sign of instability for grey solitons when the diffraction coefficients are not equal. With the diffraction coefficients set to $D_u = 1.00$ and $D_v = 1.25$, the results shown in Figure 4.7 were obtained. The optical beams are oscillating around each other with respect to depth and position. However, there is no sign of any instability in the v beam. The instability of the v beam is driven by radiation losses. However, there is no allowance for radiation loss built into the modulation equations, so no instability can occur in any of the results for the modulation equations.

Figures 4.8 – 4.9 show the evolution of grey u and v beams where the nonlocal response has been set at $\nu = 200$. The parameters for the beams have been set to $A_u = A_v = 0.8$, $B_u = B_v = 0.6$ and $U = V = 1.0$. The evolution of the beams is shown at distances of 5, 30, 60, 80, 115 and 250. The images on the left show a close in view of the beams while the right hand image shows a wide view where the x range has been extended. This is to show that the central results have not been affected by boundary reflections. The boundaries have been set at $x = -3276.8$ and $x = 3276.8$. The diffraction coefficients are $D_u = 1.00$ and $D_v = 1.25$ for all of the images. Whereas stable dark solitons oscillate around each other in position and height, the images show the increasing break up of the solitons for coupled grey nematic beams. At evolution $z = 5$ the dips of the solitons can be seen. Both the beams widen and show increasing instability as the beams evolve. The instability is affecting the u beam as much as the v beam. Both beams widen and become unstable as they evolve. This behaviour is different to the coupled beams shown in Figure 4.6 where the v beam widens and then decays, while the u beam increases in amplitude and still forms a stable soliton. It must be emphasised that the modulation equations will not show any instability until radiation loss has been built into the equations.

This chapter clearly shows the benefits of using modulation equations or an approximate model to predict some of the results obtained. Analysing the modulation equations for the position and amplitude means it is easy to see that the solutions of the solitons oscillate about each other in position and amplitude. This would not have been possible with the coupled defocusing NLS equations alone. These results can be obtained using numerical analysis of these equations. However, this is time consuming in terms of developing computer programs to produce the results and then visualising the results. The derivation of the modulation equations can be tedious. However, it is relatively straightforward mathematically. The final result of the modulation equations is that they gave much more insight into the evolution of the soliton solutions than numerical solutions alone. The oscillations can be predicted from analysis of the modulation equations and this analysis can later be confirmed using computer programs to calculate results.

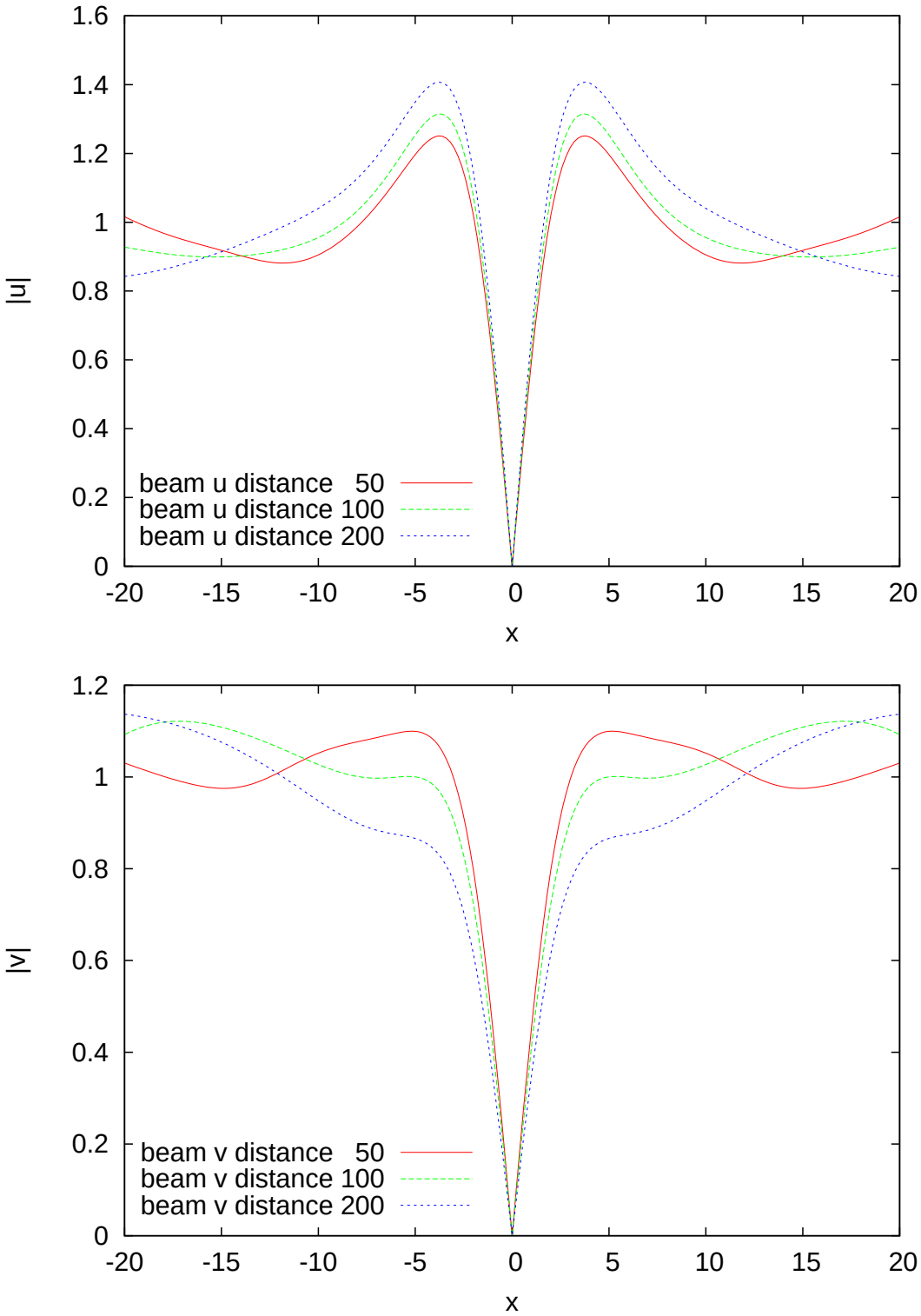

Figure 4.6: The evolution of dark nematicons calculated from the full numerical solutions of equations (4.2)–(4.4). Top image beam u, bottom image beam v. (a) solid (red) line: evolution time 50. (b) dashed (green) line: evolution time 100; (c) dotted (blue) line: evolution time 200. These are black nematicons with $A_u = A_v = 0$. Parameters are $\xi_{u0} = 1.0$, $\xi_{v0} = -1.0$, $\nu = 0.1$, $D_u = 1.00$, $D_v = 1.25$ and $L = 400$.

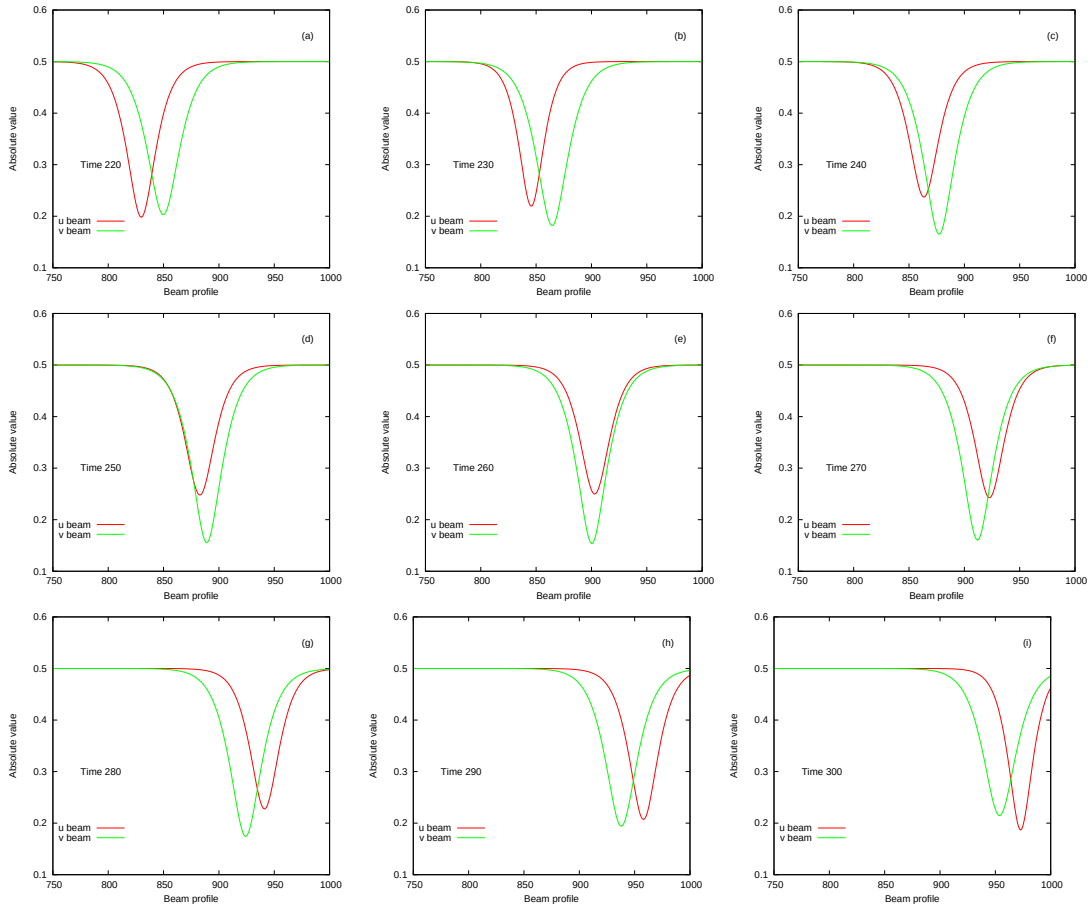

Figure 4.7: Profile of the dark optical nematicons in the x direction from solutions of the modulation equations (4.14)–(4.18). Parameters are $\xi_{u0} = 1.0$, $\xi_{v0} = -1.0$, $\nu = 200$, $D_u = 1.00$, $D_v = 1.25$. u beam solid (red) line; v beam dashed (green) line.(a) Evolution time 250 (b) Evolution time 260 (c) Evolution time 270

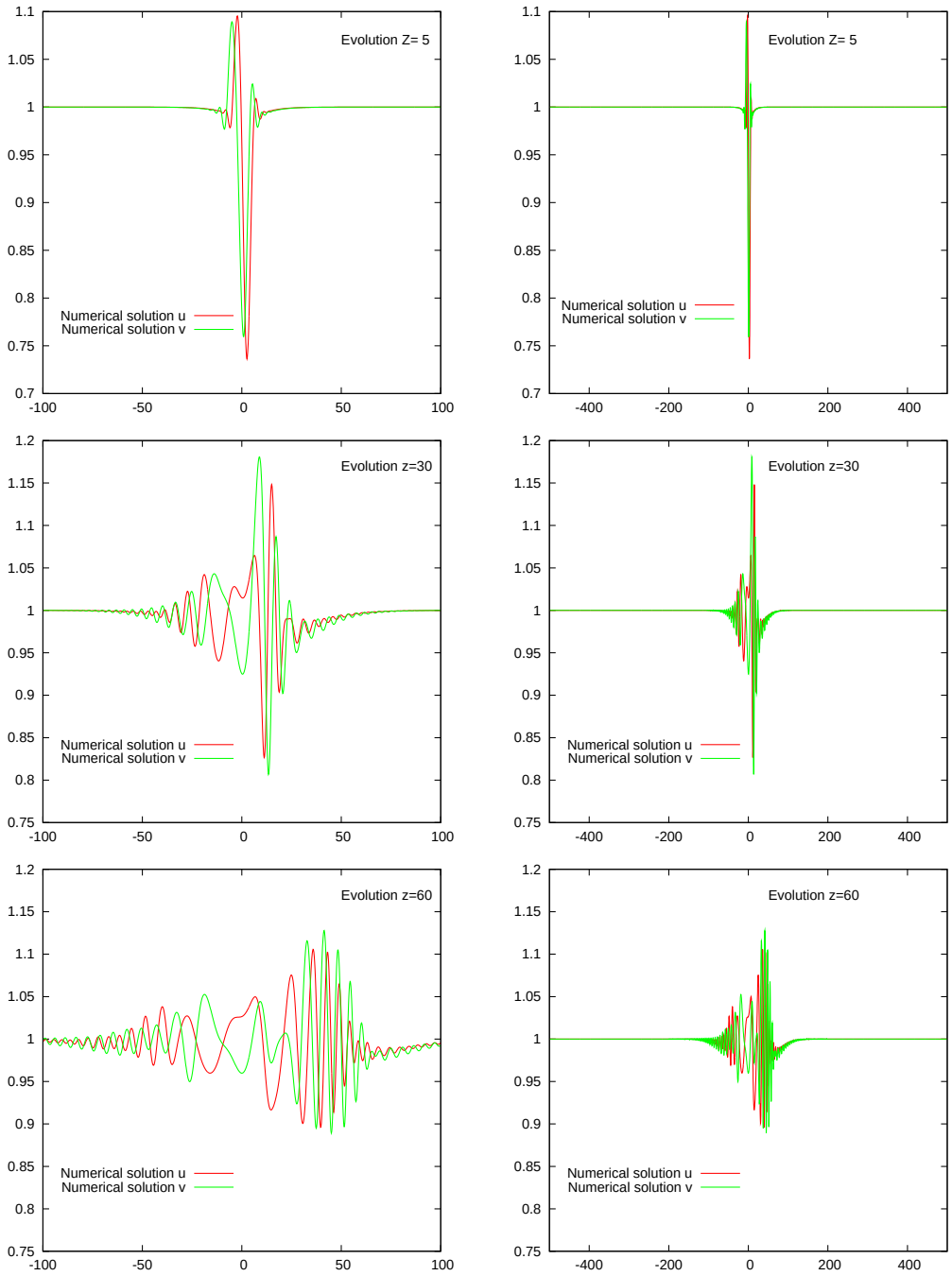

Figure 4.8: Profile of the grey optical nematicons from full numerical solutions of equations (4.2) – (4.4) for evolution distances 5, 30 and 60. The parameters used are $\xi_{u0} = 1.0$, $\xi_{v0} = -1.0$, $\nu = 200$, with diffraction coefficients of $D_u = 1.00$, $D_v = 1.25$. The images on the left are a close in view of the images on the right. The u beam is a solid (red) line; the v beam is a solid (green) line. The beams are widening and becoming increasingly unstable as they evolve. Figure continued in Figure 4.9

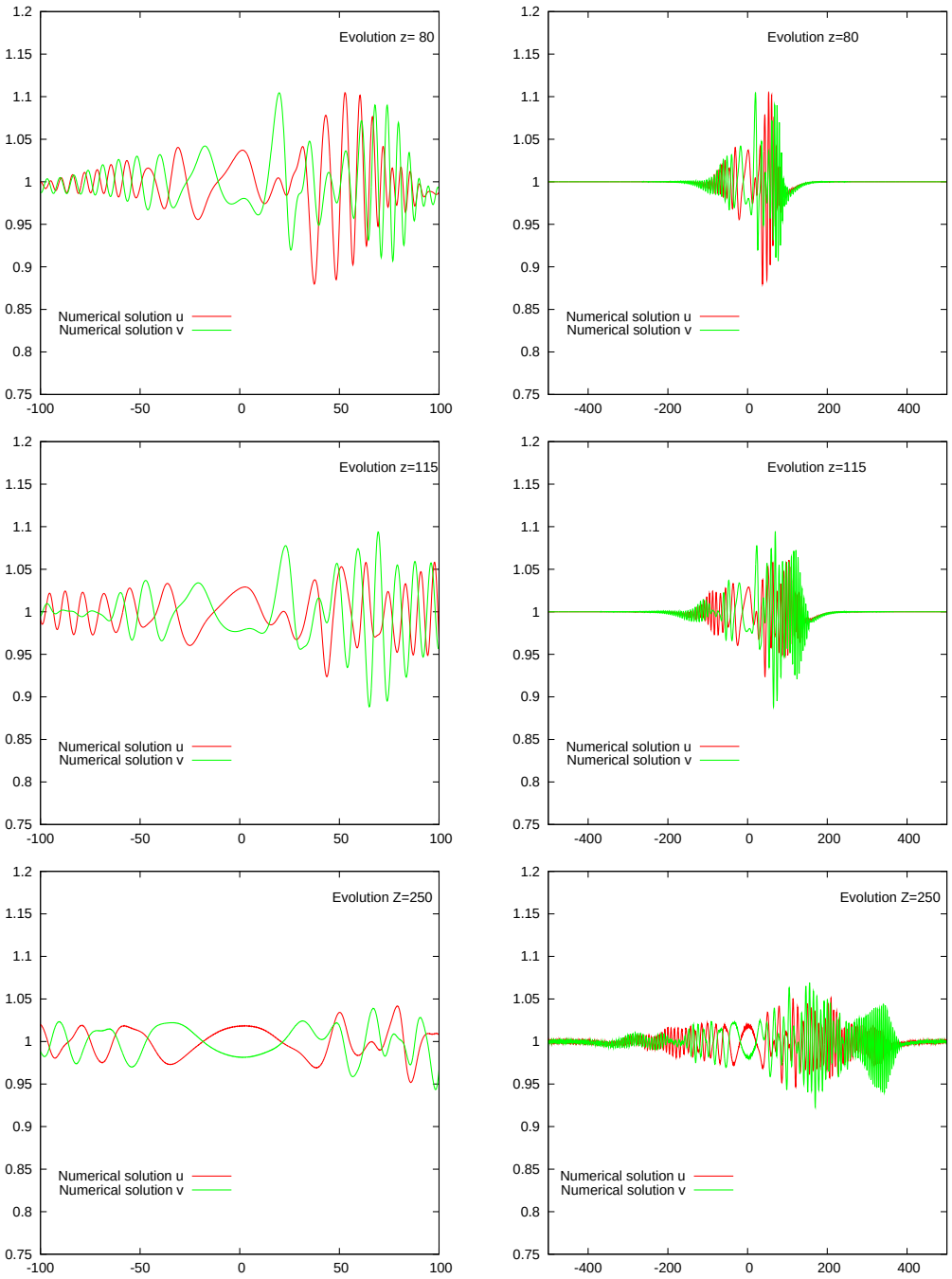

Figure 4.9: This Figure is a continuation of Figure 4.8. Profile of the grey optical nematicons from full numerical solutions of equations (4.2) – (4.4) for evolution distances 80, 115 and 250. The parameters used are $\xi_{u0} = 1.0$, $\xi_{v0} = -1.0$, $\nu = 200$, with diffraction coefficients of $D_u = 1.00$, $D_v = 1.25$. The images on the left are a close in view of the images on the right. The u beam is a solid (red) line; the v beam is a solid (green) line. The beams are widening and becoming increasingly unstable as they evolve.

Chapter 5

Conclusions

Research on nematic liquid crystals is now vast due to the usefulness of the nonlinear optical properties they possess. Their nonlinear response to light is easily controllable, making them ideal for experimental work on nonlinear optics. Passing an optical beam with appropriate parameters, such as power and width, into a nematic liquid crystal allows researchers to balance the nonlinear response, which causes the beam to self-focus, against the natural tendency of the beam to diffract. When the balance is achieved, a soliton is formed. With so much experimental work being carried out it is necessary for the mathematical understanding of nematic liquid crystals and nematicons to also progress. Some of the mathematical models that have been used to understand nematicons have been used in this thesis. In the first part of the thesis, mathematical models for the interaction of nematicons were used to enable beam on beam control so that a nematicon can be guided to a target position by another nematicon. This is a possible basis for all-optical signal control.

The nonlinear response of nematic liquid crystals relies on dipolar interactions between the elongated organic molecules and the electric field associated with propagating light beams [46, 63]. The nematic liquid crystal reorientational response to low and/or high frequency fields, together with other light-NLC interactions – for instance dye-mediated responses [123, 124] – can also be employed to spatially modulate the distribution of the molecular director. This makes nematic liquid crystal cells one of the most versatile environments to generate largely tunable refractive index profiles [44]. These methods employ external electric fields to control the distribution of the molecular director. However, the key problem in the first part of this thesis was to control the molecular director by varying the input angle of a control beam. With the correct angle of the control beam, a signal beam can be guided to a target area.

In Chapter 2 the equations that govern the interaction of two nematicons in a nematic liquid crystal were used to derive a numerical scheme that provided solutions of these equations. The same equations were also used to derive simpler equations using a variational approach. Initially, the trial function used in the simplified equations was of the form $\mathrm{sech}((x-\xi)/w)$. When the results of the simplified equations were compared with numerical solutions of the governing equations, the results were only acceptable. A different trial function of the form $\exp(-[(x-\xi)/w]^2$ was then used to derive the simplified modulation equations. The results from these simplified modulation equations when compared with results from solutions of the governing equations were again only acceptable.

In the modulation equations derived in Chapter 2 there was a conservation of mo-

mentum equation for the nematicons. In a direct analogy with classical mechanics this equation was used to treat the nematicon as a particle. The "mass" of the nematicon was defined as the power of the optical beam and the velocity was defined as the change in transverse position with propagation distance z. A key insight from Chapter 2 was that the attractive force between optical beams shown by the modulation equations was always less than the attractive force calculated from the governing equations of coupled nematicons. The simplified equations were extended by treating each nematicon as a solid rather than a particle. The results from this model then showed excellent agreement with the results obtained from numerical solutions of the governing equations. In hindsight, a key component of the interaction between the nematicons is the attractive force. When this is in good agreement with numerical results from the governing equations, then the paths of the nematicons will agree. A possible project for future research is derive better approximations to the attractive force in the simplified equations of Chapter 2. Hopefully this will produce better comparison between the results from the simplified equations and the solutions of the full governing equations.

The simplified models used in Chapters 2 and 3 did not include any allowance for diffractive radiation shed as the beam propagates. Radiation is not large for coupled nematicons in a nonlocal environment [70]. However, radiation is a key feature in the evolution of many processes involving nematicons. This is another feature that could possibly be investigated in the future. A factor against including this in a model is the added difficulty in deriving the variational equations. There is a balance between the complexity of the model and any additional benefits it might bring. In the case of coupled nematic equations in a nonlocal medium, the benefit may be small and the added complexity more trouble than it is worth.

Another possibility for further investigation is the use of more than one control beam to guide the signal beam to a target position. This would increase the options of possible target positions to guide a beam to, but also increase the complexity of the model and the programming of the beam interactions. An extra dimension in the y direction for the beam trajectories could be added. The modulation equations were derived with this in mind. However, the y variable was set to 0 for all the beams. Again, the benefits derived from the added complexity need to outweigh the difficulties involved in producing the benefits.

In Chapter 4 coupled dark nematicons were investigated. Similar techniques were used in this Chapter as were used in the other Chapters, only the details changed. The full governing equations for dark nematicons were stated and a variational formulation of these equations with suitable trial functions was used to derive the modulation equations for the coupled dark beams. There was good agreement between the results predicted by the modulation equations and the numerical solutions of the governing equations. The modulation equations showed that there should be oscillations between the beams in both position and depth leading to "merry-go-round" nematicons. This was confirmed by results from both the modulation and full numerical solutions. The comparison of the oscillations for position were better than the comparison for depth. This is due to radiation losses not being included in the modulation equations. However, the results compared quite well overall. There was no allowance for radiation or the enhancement of the attractive force between the beams due to their finite size. These might be possible to add to the modulation equations in the future to see if the comparisons for the depth improve.

Another prediction from analysis of the governing equations used in Chapter 4 was the instability of one beam when the diffraction coefficients are not equal. This was predicted in the local and the nonlocal limits. The results show the beginning of

instability in the nonlocal regime. The details of this instability need to be further investigated. This is a major extension to the work of this thesis.

Appendix A

Computer programs

The results shown in this thesis have been produced by many different computer programs. I fact before the era of personal computers in would not have been possible to have carried out the research presented in this thesis. Since the availability of public domain compilers for computers running UNIX operating system exceeds the availability of compilers running other operating systems all the programs used in this project were developed on UNIX computers. The programs were all written in the FORTRAN computer language though the version of the language varied from FORTRAN 77 to FORTRAN 2003. The earliest programs were written in FORTRAN 77 and compiled using the f77 compiler. Later programs were written in the FORTRAN 95 and FORTRAN 03 language to take advantage of improvements in language features. These programs were compiled using the GNU compiler gfortran. The listings of the programs used to calculate the results in this thesis follow.

The program used to carry out the full numerical calculations for the coupled Schrödinger equations is.

```
C.. Fourth order Runge Kutta solution of NLS with Shelly filtering
C.. Time updates are in the Fourier domain
C.. Designed for periodically forced NLS

      PROGRAM NLS4F
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      COMPLEX*16 U,CI,U0
      COMPLEX*16 U2,U20
      PARAMETER (MAX=2048,MAXY=1024)
      DIMENSION U(MAX,MAXY),TEMP(MAX,MAXY)
      DIMENSION U0(MAX,MAXY),UM(MAX,MAXY),ddx(MAX)
      DIMENSION UMM(MAX)
      DIMENSION U20(MAX,MAXY),UM2(MAX,MAXY),UMM2(MAX)
      DIMENSION U2(MAX,MAXY)
      DOUBLE PRECISION NDP1,NDP2,Wrk1,Wrk2,Wrk3
      INTEGER n2,iter,Wrk4

      COMMON/INTEGERS/N,NY,NSTEPS,IMID,IDEBUG,NSX,NSY
      COMMON/TIME/DT,T0,TF,SRF
      COMMON/LENGTH/SPACE,SPACEY
      COMMON/AMP/A1,A2,W1,W2,R,EPS,OM,XD,YD,V,XD2,YD2,V2,xmin,
     &      xmax,ymin,ymax,TargetX
      COMMON/PHASE/phi
      COMMON/NEWPAR/rnu,th0,te00,te01,qs,diff1,diff2,AAAA,BBBB
      COMMON/CONTINUE/NCONT
      COMMON/ASORB/NDP1,NDP2,DMP1,DMP2
      COMMON/SOLN/U,U0,U2,U20,TEMP
      COMMON/DERIV/ddx

C..CALLING ROUTINES TO BEGIN THE CALCULATIONS
```

```
      call Wtime('Start')

      CALL SETUP
      CALL INITIALIZE
      iter=1

383 Continue

C..WRITE THE SOLUTION TO THE SURFER FILE FOR THE FIRST TIME
c      CALL OUTPUT

      CI=(0.D0,1.D0)
      dx=space/dble(n)
      dy=spacey/dble(NY)
      xpos=XD
      ypos=YD
      xpos2=XD2
      ypos2=YD2

      CALL HEAT

C..THE MAIN BODY OF THE PROGRAM

      OPEN (90,FILE='nlsheatmax.dat',STATUS='UNKNOWN')
      CLOSE (90,STATUS='DELETE')
      OPEN (90,FILE='nlsheatmax.dat',STATUS='UNKNOWN')

      OPEN (80,FILE='nlsheatoutmid.dat',STATUS='UNKNOWN')
      CLOSE (80,STATUS='DELETE')
      OPEN (80,FILE='nlsheatoutmid.dat',STATUS='UNKNOWN')

      OPEN (82,FILE='heatoutmid.dat',STATUS='UNKNOWN')
      CLOSE (82,STATUS='DELETE')
      OPEN (82,FILE='heatoutmid.dat',STATUS='UNKNOWN')

      OPEN (86,FILE='evolve.dat',STATUS='UNKNOWN')
      CLOSE (86,STATUS='DELETE')
      OPEN (86,FILE='evolve.dat',STATUS='UNKNOWN')

      OPEN (91,FILE='nlsheatmax2.dat',STATUS='UNKNOWN')
      CLOSE (91,STATUS='DELETE')
      OPEN (91,FILE='nlsheatmax2.dat',STATUS='UNKNOWN')

      OPEN (81,FILE='nlsheatoutmid2.dat',STATUS='UNKNOWN')
      CLOSE (81,STATUS='DELETE')
      OPEN (81,FILE='nlsheatoutmid2.dat',STATUS='UNKNOWN')

      OPEN (87,FILE='evolve2.dat',STATUS='UNKNOWN')
      CLOSE (87,STATUS='DELETE')
      OPEN (87,FILE='evolve2.dat',STATUS='UNKNOWN')

      IP=AINT(DBLE(NSTEPS)/SRF+0.5)
      IF (IP .EQ. 0) THEN IP = 1

      do 12 j=1,ny
         do 11 i=1,n
            um(i,j)=abs(u(i,j))
            um2(i,j)=abs(u2(i,j))
 11      continue
 12   continue

      umax=0.0d0
      umax2=0.0d0
      do 33 i=1,n
            if(um(i,NY/2+1).gt.umax) then
               umax=um(i,NY/2+1)
               imax=i
            end if
            if(um2(i,NY/2+1).gt.umax2) then
               umax2=um2(i,NY/2+1)
               imax2=i
            end if
  33  continue
```

```
      DO 99 I=1,N
         umm(I)=um(i,NY/2+1)
         umm2(I)=um2(i,NY/2+1)
99    CONTINUE

      n2=N/2
      xpos=xpos+space/2.0d0
      call maxfind(umm,umax,xpos,dx,n)

      xpos=xpos-space/2.0d0
      ipos=aint(xpos/dx+0.5)+1

      xpos2=xpos2+space/2.0d0
      call maxfind(umm2,umax2,xpos2,dx,n)

      xpos2=xpos2-space/2.0d0
      ipos2=aint(xpos2/dx+0.5)+1

      IF (IDEBUG .EQ. 1) THEN
        WRITE (6,*)
        WRITE (6,800)
800     FORMAT (4X, ' STEP # ',5X,'% DONE',10X,'TIME',
     &   16X,'MIDPOINT MAGNITUDE')
        WRITE (6,900) 0,0,T0,umax,xpos,umax2,xpos2,
     &       temp(N/2+1,NY/2+1)
      ENDIF

      te1=xmin*dble(N)/SPACE+dble(N)/2.0d0+1.0d0
      te2=xmax*dble(N)/SPACE+dble(N)/2.0d0+1.0d0

      Wrk1=space/Dble(N)
      Wrk2=.5D0*space
      Wrk4=NY/2+1
      DO 767 I=AINT(te1),AINT(te2),NSX
         Wrk3=Wrk1*(I-1)-Wrk2
         WRITE (86,100) Wrk3,0.0d0,ABS(U(I,Wrk4))
         WRITE (87,100) Wrk3,0.0d0,ABS(U2(I,Wrk4))
767   CONTINUE
      write (86,101)
      write (87,101)

      WRITE (90,910) T0,umax,xpos
      WRITE (91,910) T0,umax2,xpos2
      DO 10 J=1,NSTEPS
         T = (J-1)*DT+T0
         CALL RUNGE_KUTTA
         CALL HEAT

         IF (IDebug.eq.1) then
            write(6,*) '|U(N/2,NY/2+1)|=', ABS(U(N/2,NY/2+1))
            write(6,*) '|U2(N/2,NY/2+1)|=', ABS(U2(N/2,NY/2+1))
         EndIf

         IF (MOD(J,IMID).EQ.0) THEN
           do 21 i=1,n
              do 22 jj=1,ny
                 um(i,jj)=abs(u(i,jj))
                 um2(i,jj)=abs(u2(i,jj))
22            continue
21         continue
           umax=0.0d0
           do 37 i=1,n
              if(um(i,NY/2+1).gt.umax) then
                 umax=um(i,NY/2+1)
                 imax=i
              end if
 37        continue

           umax2=0.0d0
           do 38 i=1,n
               if(um2(i,NY/2+1).gt.umax2) then
                  umax2=um2(i,NY/2+1)
                  imax2=i
               end if
```

```
 38       continue

          DO 919 I=1,N
             umm(I)=um(i,NY/2+1)
             umm2(i)=um2(I,NY/2+1)
919       CONTINUE

          xpos=xpos+space/2.0d0
          call maxfind(umm,umax,xpos,dx,n)
          xpos=xpos-space/2.0d0
          ipos=aint(xpos/dx+0.5)+1

          xpos2=xpos2+space/2.0d0
          call maxfind(umm2,umax2,xpos2,dx,n)
          xpos2=xpos2-space/2.0d0
          ipos2=aint(xpos2/dx+0.5)+1

          IF (IDEBUG .EQ. 1) THEN
              WRITE (6,900) J,FLOAT(J)/FLOAT(NSTEPS)*100,T+DT,umax,
     &                  xpos,umax2,xpos2,temp(N/2+1,NY/2+1)
          ENDIF
          WRITE (90,910) T+DT,umax,xpos
          WRITE (91,910) T+DT,umax2,xpos2

          te1=xmin*dble(N)/SPACE+dble(N)/2.0d0+1.0d0
          te2=xmax*dble(N)/SPACE+dble(N)/2.0d0+1.0d0

          Wrk1=space/Dble(N)
          Wrk2=.5D0*space
          Wrk4=NY/2+1
          DO 766 I=AINT(te1),AINT(te2),NSX
             Wrk3=Wrk1*(I-1)-Wrk2
             WRITE (86,100) Wrk3,T,ABS(U(I,Wrk4))
             WRITE (87,100) Wrk3,T,ABS(U2(I,Wrk4))
766       CONTINUE
          write (86,101)
          write (87,101)

        ENDIF
900     FORMAT (1X,I8,2X,F12.2,2X,F16.6,2X,5F28.12)
910     FORMAT (1X,F16.6,2X,4F28.12)
c        IF (MOD(J,IP) .EQ. 0) THEN
c           CALL OUTPUT
c        ENDIF
 10   CONTINUE

!
!     If the absolute error between the X position calculated and the target position is greater than 1D-5
!     then call the Predict function to calculate the new angle. The PrimRad function ensures that the angle is
!     between Pi/2 and -Pi/2. The program has 20 chances to do this. If the laser beam does not form a soliton
!     or the two beams are very close then it may not be possible to achieve the target value of X.
!
       WRITE (6,1300) iter,xpos,v,dabs(TargetX-xpos)
if ((iter .le. 20).and.(dabs(TargetX-xpos)>1.0D-5)) then
               v=PrimRad(Predict(xpos,v,iter,TargetX,xpos2,M_cross,M_Bnd))
iter=iter+1
        call initializ2
goto 383
endif

C..WRITING THE FINAL SOLUTION TO A GRAPHER FILE

      OPEN (30,FILE='nlsheatout.dat',STATUS='UNKNOWN')
      CLOSE (30,STATUS='DELETE')
      OPEN (30,FILE='nlsheatout.dat',STATUS='UNKNOWN')

      OPEN (32,FILE='heatout.dat',STATUS='UNKNOWN')
      CLOSE (32,STATUS='DELETE')
      OPEN (32,FILE='heatout.dat',STATUS='UNKNOWN')

      OPEN (44,FILE='nlsheatout2.dat',STATUS='UNKNOWN')
      CLOSE (44,STATUS='DELETE')
      OPEN (44,FILE='nlsheatout2.dat',STATUS='UNKNOWN')
```

```
      dxx=SPACE/DBLE(N)
      te1=xmin*dble(N)/SPACE+dble(N)/2.0d0+1.0d0
      te2=xmax*dble(N)/SPACE+dble(N)/2.0d0+1.0d0
      te1y=ymin*dble(NY)/SPACEY+dble(NY)/2.0d0+1.0d0
      te2y=ymax*dble(NY)/SPACEY+dble(NY)/2.0d0+1.0d0
      DO 60 K=AINT(te1y),AINT(te2y),NSY
         yy=SPACEY*(DBLE(K-1)/DBLE(NY))-SPACEY/2.DO
         Wrk1=space/Dble(N)
         Wrk2=.5D0*space
         Wrk4=NY/2+1
         DO 50 I=AINT(te1),AINT(te2),NSX
            Wrk3=Wrk1*(I-1)-Wrk2
            WRITE (30,100) Wrk3,YY,ABS(U(I,K))
            WRITE (44,100) Wrk3,YY,ABS(U2(I,K))
            WRITE (32,100) Wrk3,YY,TEMP(I,K)
 50      CONTINUE
         write (30,101)
         write (44,101)
         write (32,101)
 60   CONTINUE


      Wrk1=space/Dble(N)
      Wrk2=.5D0*space
      Wrk4=NY/2+1
      DO 555 I=1,N
         Wrk3=Wrk1*(I-1)-Wrk2
        WRITE (80,100) Wrk3,ABS(U(I,Wrk4))
        WRITE (81,100) Wrk3,ABS(U2(I,Wrk4))
        WRITE (82,100) Wrk3,TEMP(I,Wrk4)
 555  CONTINUE

      CLOSE (30)
      CLOSE (32)
      CLOSE (44)
      CLOSE (50)
      CLOSE (80)
      CLOSE (81)
      CLOSE (82)

      call Wtime('End')

 100  FORMAT (3F28.16)
 101  FORMAT()
1000  Format(A20, i2.2, ':', i2.2, ':', i2.2)
1300  Format(I4,'  Xpos',F16.8,'  V',F16.8,'  Abs error',F16.8)

      END
*
**********************************************************************
*
      SUBROUTINE SETUP
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DOUBLE PRECISION NDP1,NDP2

      COMMON/INTEGERS/N,NY,NSTEPS,IMID,IDEBUG,NSX,NSY
      COMMON/TIME/DT,TO,TF,SRF
      COMMON/LENGTH/SPACE,SPACEY
      COMMON/AMP/A1,A2,W1,W2,R,EPS,OM,XD,YD,V,XD2,YD2,V2,
     &      xmin,xmax,ymin,ymax,TargetX
      COMMON/PHASE/phi
      COMMON/NEWPAR/rnu,th0,te00,te01,qs,diff1,diff2,AAAA,BBBB
      COMMON/CONTINUE/NCONT
      COMMON/ASORB/NDP1,NDP2,DMP1,DMP2

      OPEN (15,FILE='nlsheat2d.in',STATUS='OLD')
      REWIND (15)
C..   N = number of x spatial points
      READ (15,200) N
C..   NY = number of y spatial points
      READ (15,200) NY
c..   dt = timestep
      READ (15,220) DT
c..   t0 = start time
```

```
      READ (15,220) T0
c..   tf = final time
      READ (15,220) TF
c..   nsteps = number of time steps to take
      NSTEPS = AINT((TF-T0)/DT+0.5)
C..   space = x interval
      READ (15,220) SPACE
C..   spacey = y interval
      READ (15,220) SPACEY
C..   NSX =  x spacing for plotting 3D graph
      READ (15,200) NSX
C..   NSY =  y spacing for plotting 3D graph
      READ (15,200) NSY
c..   xmin = minimum x point for plot
      READ (15,220) xmin
c..   xmax = maximum x point for plot
      READ (15,220) xmax
c..   ymin = minimum y point for plot
      READ (15,220) ymin
c..   ymax = maximum y point for plot
      READ (15,220) ymax
c..   a1 = initial amplitude for u
      READ (15,220) A1
c..   a2 = initial amplitude for v
      READ (15,220) A2
c..   w1 = inital width for u
      READ (15,220) W1
c..   w2 = inital width for v
      READ (15,220) W2
c..   phi = initial phase difference
      READ (15,220) phi
c..   xd = x position for u
      READ (15,220) XD
c..   yd = y offset position for u
      READ (15,220) YD
c..   V = velocity for u
      READ (15,220) V
c..   xd2 = x position for v
      READ (15,220) XD2
c..   yd2 = y offset position for v
      READ (15,220) YD2
c..   V2 = velocity for v
      READ (15,220) V2
c..   diff1 = diffraction coefficient for u
      READ (15,220) diff1
c..   diff2 = diffraction coefficient for v
      READ (15,220) diff2
c.    AAAA = A for u
      READ (15,220) AAAA
c.    BBBB = B for u
      READ (15,220) BBBB
c..   rnu = nu
      READ (15,220) rnu
c..   th0 = theta_{0}
      READ (15,220) th0
      pi=dacos(-1.d0)
      te00=pi/4.d0
      te01=pi/4.d0
c..   qs = q_{s}
      READ (15,220) qs
c..   r = rotating phase factor
      READ (15,220) R
c..   eps = amplitude of periodic forcing
      READ (15,220) EPS
c..   om = frequency of periodic forcing, omega
      READ (15,220) OM
c..   x distance from end for start of damping's polynomial cutoff
      READ (15,220) NDP1
c..   x distance from end for end of damping's polynomial cutoff
      READ (15,220) NDP2
c..   height of damping profile
      READ (15,220) DMP1
c..   coefficient of x inside of damping profile (1/width)
      READ (15,220) DMP2
```

```
c..   plot frequency for midpoint data
      READ (15,200) IMID
c..   number of surfer plots to write out
      READ (15,220) SRF
c..   flag to decide to plot damping or not
      READ (15,200) IDEBUG
c..   ncont = 0 for initial start, 1 for continuing from last output file
      READ (15,200) NCONT
c..   Target X position. Want to get a solution that ends at this X position
      READ (15,220) TargetX
      CLOSE (15)

c      IF (IDEBUG .EQ. 1) THEN
       WRITE (6,*)
       WRITE (6,240) N, '... number of x spatial points'
       WRITE (6,240) NY, '... number of y spatial points'
       WRITE (6,260) DT, '... time step'
       WRITE (6,260) T0, '... start time'
       WRITE (6,260) TF, '... stop time'
       WRITE (6,240) NSTEPS, '... number of Runge-Kutta steps'
       WRITE (6,260) SPACE, '... x spatial interval'
       WRITE (6,260) SPACEY, '... y spatial interval'
       WRITE (6,260) xmin, '... minimum x point for plot'
       WRITE (6,260) xmax, '... maximum x point for plot'
       WRITE (6,260) ymin, '... minimum y point for plot'
       WRITE (6,260) ymax, '... maximum y point for plot'
       IF (NCONT .EQ. 0) THEN
           WRITE (6,260) A1, '... initial amplitude for u'
           WRITE (6,260) W1, '... initial width for u'
           WRITE (6,260) A2, '... initial amplitude for v'
           WRITE (6,260) W2, '... initial width for v'
           WRITE (6,260) XD, '... x centre position for u'
           WRITE (6,260) YD, '... y centre position for u'
           WRITE (6,260) V, '... velocity for u'
           WRITE (6,260) XD2, '... x centre position for v'
           WRITE (6,260) YD2, '... y centre position for v'
           WRITE (6,260) V2, '... velocity for v'
           WRITE (6,260) diff1,'... diffraction coefficient for u'
           WRITE (6,260) AAAA,'A for u'
           WRITE (6,260) diff2,'... diffraction coefficient for v'
           WRITE (6,260) BBBB,'B for v'
           WRITE (6,260) rnu, '... nu'
           WRITE (6,260) eps,'... epsilon'
           WRITE (6,260) te00, '... te00'
           WRITE (6,260) te01, '... te01'
           WRITE (6,260) qs, '... q_{s}'
       ENDIF
       WRITE (6,260) EPS, '... periodic forcing amplitude'
       WRITE (6,260) OM, '... periodic forcing frequency'
       WRITE (6,260) NDP1, '... start of polynomial damping cutoff'
       WRITE (6,260) NDP2, '... end of polynomial damping cutoff'
       WRITE (6,260) DMP1, '... damping layer amplitude'
       WRITE (6,260) DMP2, '... damping layer decay rate'
       WRITE (6,260) SRF, '... number of surfer profiles'
       WRITE (6,240) IDEBUG, '... debug parameter (1=yes, 0=no)'
       WRITE (6,240) NCONT, '... continuation parameter (1=yes, 0=no)'
       Write (6,260) TargetX, '...target value of X'

 200  FORMAT (I10,14X,I6)
 220  FORMAT (F20.16,4X,I6)
 240  FORMAT (8X,I10,12X,A40)
 260  FORMAT (8X,F24.16,2X,A40)

      END

*
**********************************************************************
*
      SUBROUTINE INITIALIZE
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DOUBLE PRECISION NDP1,NDP2,Wrk1,Wrk2,Wrk3
      COMPLEX*16 CI,U,INTF,INTB,U0,U2,U20,INTF2,INTB2
      PARAMETER (MAX=2048,MAXY=1024)
      DIMENSION U(MAX,MAXY),INTF(MAX),INTB(MAX),TEMP(MAX,MAXY)
```

```
      DIMENSION INTF2(MAX),INTB2(MAX)
c      DIMENSION X1(MAX)
      DIMENSION DM(MAX),DMTMP(MAX),U0(MAX,MAXY)
      DIMENSION U2(MAX,MAXY),U20(MAX,MAXY)

      COMMON/INTEGERS/N,NY,NSTEPS,IMID,IDEBUG,NSX,NSY
      COMMON/TIME/DT,T0,TF,SRF
      COMMON/LENGTH/SPACE,SPACEY
      COMMON/AMP/A1,A2,W1,W2,R,EPS,OM,XD,YD,V,XD2,YD2,V2,
     &      xmin,xmax,ymin,ymax,TargetX
      COMMON/PHASE/phi
      COMMON/NEWPAR/rnu,th0,te00,te01,qs,diff1,diff2,AAAA,BBBB
      COMMON/CONTINUE/NCONT
      COMMON/ASORB/NDP1,NDP2,DMP1,DMP2
      COMMON/SOLN/U,U0,U2,U20,TEMP

      COMMON/INT/INTF,INTB,INTF2,INTB2
      COMMON/DAMP/DM

      CI=(0.D0,1.D0)
      pi=dacos(-1.0d0)

C..CREATING THE INITIAL PROFILE AND WRITING IT TO A GRAPHER FILE

      IF (NCONT .EQ. 0) THEN

      OPEN (10,FILE='nlsheatin.dat',STATUS='UNKNOWN')
      CLOSE (10,STATUS='DELETE')
      OPEN (10,FILE='nlsheatin.dat',STATUS='UNKNOWN')
      OPEN (11,FILE='nlsheatin2.dat',STATUS='UNKNOWN')
      CLOSE (11,STATUS='DELETE')
      OPEN (11,FILE='nlsheatin2.dat',STATUS='UNKNOWN')
      OPEN (56,FILE='heatin.dat',STATUS='UNKNOWN')
      CLOSE (56,STATUS='DELETE')
      OPEN (56,FILE='heatin.dat',STATUS='UNKNOWN')

      OPEN (85,FILE='nlsheatmidin.dat',STATUS='UNKNOWN')
      CLOSE (85,STATUS='DELETE')
      OPEN (85,FILE='nlsheatmidin.dat',STATUS='UNKNOWN')
      OPEN (86,FILE='heatmidin.dat',STATUS='UNKNOWN')
      CLOSE (86,STATUS='DELETE')
      OPEN (86,FILE='heatmidin.dat',STATUS='UNKNOWN')

      OPEN (87,FILE='nlsheatmidin2.dat',STATUS='UNKNOWN')
      CLOSE (87,STATUS='DELETE')
      OPEN (87,FILE='nlsheatmidin2.dat',STATUS='UNKNOWN')

      DO 21 K=1,NY
            YY=SPACEY*(DBLE(K-1)/DBLE(NY))-SPACEY/2.D0
    Wrk1=space/Dble(N)
    Wrk2=.5D0*space
    DO 20 I=1,N
       Wrk3=Wrk1*(I-1)-Wrk2
       rr1=dsqrt((Wrk3-XD)*(Wrk3-XD)+(YY-YD)*(YY-YD))
       arg1=V*(Wrk3-XD)
       U(I,K)=A1*EXP(CI*arg1)/COSH(rr1/W1)
       rr1=dsqrt((Wrk3-XD2)*(Wrk3-XD2)+(YY-YD2)*(YY-YD2))
       arg1=V2*(Wrk3-XD2)
       U2(I,K)=A2*EXP(CI*arg1)/COSH(rr1/W2)
               U0(I,K)=U(I,K)
       U20(I,K)=U2(I,K)
 20        CONTINUE
 21   CONTINUE
      DO 666 K=1,NY
         YY=SPACEY*(DBLE(K-1)/DBLE(NY))-SPACEY/2.D0
         do 665 i=1,n
            X=SPACE*(DBLE(I-1)/DBLE(N))-SPACE/2.D0
            temp(i,k)=0.0d0
 665     continue
 666  continue

      dx=space/dble(n)
      dxx=SPACE/DBLE(N)
      te1=xmin*dble(N)/SPACE+dble(N)/2.0d0+1.0d0
```

```
      te2=xmax*dble(N)/SPACE+dble(N)/2.0d0+1.0d0
      te1y=ymin*dble(NY)/SPACEY+dble(NY)/2.0d0+1.0d0
      te2y=ymax*dble(NY)/SPACEY+dble(NY)/2.0d0+1.0d0
      do 300 K=AINT(te1y),AINT(te2y),NSY
         yy=SPACEY*(DBLE(K-1)/DBLE(NY))-SPACEY/2.D0
         do 350 I=AINT(te1),AINT(te2),NSX
            XX=SPACE*(DBLE(I-1)/DBLE(N))-SPACE/2.D0
            write(10,351) xx,yy,abs(U(i,k))
            write(11,351) xx,yy,abs(U2(i,k))
            write(56,351) xx,yy,temp(i,k)
  350    continue
         write(10,352)
         write(11,352)
         write(56,352)
  300 continue
  351 format(3F28.16)
  352 format()


      Wrk1=space/Dble(N)
      Wrk2=.5D0*space
      Wrk4=NY/2+1
      DO 555 I=1,N
         Wrk3=Wrk1*(I-1)-Wrk2
         WRITE (85,351) Wrk3,ABS(U(I,Wrk4))
         WRITE (87,351) Wrk3,ABS(U2(I,Wrk4))
         WRITE (86,351) Wrk3,temp(i,Wrk4)
 555  CONTINUE

      ELSE

      OPEN (10,FILE='nlsheatout.dat',STATUS='OLD')
      OPEN (11,FILE='nlsheatout2.dat',STATUS='OLD')
      REWIND (10)
      REWIND (11)
      CLOSE (10)

      ENDIF


C..CREATING THE EDGE-DAMPING FUNCTION

      DM(1)=DMP1
      DMTMP(1)=DMP1
C..EDGE TO START OF DAMPING LAYER
      DO 50 I=2,AINT(NDP1*N/SPACE)+1
         X2=SPACE*(DBLE(I-1)/DBLE(N))
         DMTMP(I)=DMP1/(COSH(DMP2*X2))**2.D0
         DM(I)=DMTMP(I)
         DM(N+2-I)=DM(I)
         DMTMP(N+2-I)=DMTMP(I)
 50   CONTINUE
C..DAMPING LAYER
      DO 60 I=AINT(NDP1*N/SPACE)+2,AINT(NDP2*N/SPACE)+1
         X=1.D0-((I-1)*SPACE/N-NDP1)/(NDP2-NDP1)
         F=(126.D0-420.D0*X+540.D0*X**2.D0-315.D0*X**3.D0
     &             +70.D0*X**4.D0)*X**5.D0
         X2=SPACE*(DBLE(I-1)/DBLE(N))
         DMTMP(I)=DMP1/(COSH(DMP2*X2))**2.D0
         DM(I)=F*DMTMP(I)
         DMTMP(N+2-I)=DMTMP(I)
         DM(N+2-I)=DM(I)
 60   CONTINUE
C..INTERIOR LAYER
      DO 70 I=AINT(NDP2*N/SPACE)+2,N/2+1
         X2=SPACE*(DBLE(I-1)/DBLE(N))
         DMTMP(I)=DMP1/(COSH(DMP2*X2))**2.D0
         DM(I)=0.D0
         DMTMP(N+2-I)=DMTMP(I)
         DM(N+2-I)=DM(I)
 70   CONTINUE

      IF (IDEBUG .EQ. 1) THEN
      OPEN (40,FILE='damp.dat',STATUS='UNKNOWN')
```

```
      CLOSE (40,STATUS='DELETE')
      OPEN (40,FILE='damp.dat',STATUS='UNKNOWN')

      DO 80 I=1,N
         X2=SPACE*(DBLE(I-1)/DBLE(N))-SPACE/2.D0
         WRITE (40,100) X2,DM(I),DMTMP(I)
 80   CONTINUE
      WRITE (40,100) SPACE/2.0,DM(1),DMTMP(1)
      CLOSE (40)
      ENDIF

C..INITIALIZING THE FOURIER TRANSFORM AND INTEGRATING FACTORS
      PI=ACOS(-1.D0)
      CI=(0.D0,1.D0)
      CALL FFTI(N)

C..FORWARD INTEGRATING FACTOR
      DO 90 K=2,N/2+1
         AK=2.D0*PI/SPACE*DBLE(K-1)
         INTF(K)=EXP(CI*diff1*(AK**2.D0/2.D0)*DT/2.D0)
         INTB(K)=1.D0/INTF(K)
         INTF(N+2-K)=EXP(CI*diff1*(AK**2.D0/2.D0)*DT/2.D0)
         INTB(N+2-K)=1.D0/INTF(N+2-K)
         INTF2(K)=EXP(CI*diff2*(AK**2.D0/2.D0)*DT/2.D0)
         INTB2(K)=1.D0/INTF2(K)
         INTF2(N+2-K)=EXP(CI*diff2*(AK**2.D0/2.D0)*DT/2.D0)
         INTB2(N+2-K)=1.D0/INTF2(N+2-K)
 90   CONTINUE
      INTF(1)=1.0D0
      INTB(1)=1.0D0
      INTF2(1)=1.0D0
      INTB2(1)=1.0D0

C.. Write surfer headers

 100  FORMAT (' ',T2,F10.4,TR5,3F28.16)
 505  FORMAT (A4)
 507  FORMAT (I7,TR5,I7)
*
      END
*
**********************************************************************
*
      SUBROUTINE OUTPUT
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      COMPLEX*16 U,U0,U2,U20
      PARAMETER (MAX=2048,MAXY=1024)
      DIMENSION U(MAX,MAXY),TEMP(MAX,MAXY)
      DIMENSION U0(MAX,MAXY)
      DIMENSION U2(MAX,MAXY),U20(MAX,MAXY)

      COMMON/INTEGERS/N,NY,NSTEPS,IMID,IDEBUG,NSX,NSY
      COMMON/TIME/DT,T0,TF,SRF
      COMMON/SOLN/U,U0,U2,U20,TEMP
      COMMON/LENGTH/SPACE,SPACEY
      COMMON/AMP/A1,A2,W1,W2,R,EPS,OM,XD,YD,V,XD2,YD2,V2,
     &      xmin,xmax,ymin,ymax,TargetX
      COMMON/NEWPAR/rnu,th0,te00,te01,qs,diff1,diff2,AAAA,BBBB
*
 500  FORMAT (F28.16)
      END
*
**********************************************************************
*

      SUBROUTINE INITIALIZ2
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DOUBLE PRECISION NDP1,NDP2,Wrk1,Wrk2,Wrk3
      COMPLEX*16 CI,U,INTF,INTB,U0,U2,U20,INTF2,INTB2
      PARAMETER (MAX=2048,MAXY=1024)
      DIMENSION U(MAX,MAXY),INTF(MAX),INTB(MAX),TEMP(MAX,MAXY)
      DIMENSION INTF2(MAX),INTB2(MAX)
c      DIMENSION X1(MAX)
      DIMENSION DM(MAX),DMTMP(MAX),U0(MAX,MAXY)
```

```
      DIMENSION U2(MAX,MAXY),U20(MAX,MAXY)

      COMMON/INTEGERS/N,NY,NSTEPS,IMID,IDEBUG,NSX,NSY
      COMMON/TIME/DT,T0,TF,SRF
      COMMON/LENGTH/SPACE,SPACEY
      COMMON/AMP/A1,A2,W1,W2,R,EPS,OM,XD,YD,V,XD2,YD2,V2,
     &      xmin,xmax,ymin,ymax,TargetX
      COMMON/PHASE/phi
      COMMON/NEWPAR/rnu,th0,te00,te01,qs,diff1,diff2,AAAA,BBBB
      COMMON/CONTINUE/NCONT
      COMMON/ASORB/NDP1,NDP2,DMP1,DMP2
      COMMON/SOLN/U,U0,U2,U20,TEMP

      COMMON/INT/INTF,INTB,INTF2,INTB2
      COMMON/DAMP/DM

      CI=(0.D0,1.D0)
      pi=dacos(-1.0d0)

C..CREATING THE INITIAL PROFILE AND WRITING IT TO A GRAPHER FILE

      IF (NCONT .EQ. 0) THEN

         DO K=1,NY
            YY=SPACEY*(DBLE(K-1)/DBLE(NY))-SPACEY/2.D0
    Wrk1=space/Dble(N)
    Wrk2=.5D0*space
    DO I=1,N
       Wrk3=Wrk1*(I-1)-Wrk2
       rr1=dsqrt((Wrk3-XD)*(Wrk3-XD)+(YY-YD)*(YY-YD))
       arg1=V*(Wrk3-XD)
       U(I,K)=A1*EXP(CI*arg1)/COSH(rr1/W1)
       rr1=dsqrt((Wrk3-XD2)*(Wrk3-XD2)+(YY-YD2)*(YY-YD2))
       arg1=V2*(Wrk3-XD2)
       U2(I,K)=A2*EXP(CI*arg1)/COSH(rr1/W2)
       U0(I,K)=U(I,K)
       U20(I,K)=U2(I,K)
            enddo
         enddo

         DO K=1,NY
            YY=SPACEY*(DBLE(K-1)/DBLE(NY))-SPACEY/2.D0
            do i=1,n
               X=SPACE*(DBLE(I-1)/DBLE(N))-SPACE/2.D0
               temp(i,k)=0.0d0
            enddo
         enddo

         dx=space/dble(n)
         dxx=SPACE/DBLE(N)
         te1=xmin*dble(N)/SPACE+dble(N)/2.0d0+1.0d0
         te2=xmax*dble(N)/SPACE+dble(N)/2.0d0+1.0d0
         te1y=ymin*dble(NY)/SPACEY+dble(NY)/2.0d0+1.0d0
         te2y=ymax*dble(NY)/SPACEY+dble(NY)/2.0d0+1.0d0

         var3=aint(te1y)
         var4=aint(te2y)
         do K= var3, var4, NSY
            yy=SPACEY*(DBLE(K-1)/DBLE(NY))-SPACEY/2.D0
            var1=aint(te1)
            var2=aint(te2)
            do I= var1, var2, NSX
               XX=SPACE*(DBLE(I-1)/DBLE(N))-SPACE/2.D0
            enddo
         enddo
      ENDIF

C..CREATING THE EDGE-DAMPING FUNCTION

      DM(1)=DMP1
      DMTMP(1)=DMP1
C..EDGE TO START OF DAMPING LAYER

      var5=aint(ndp1*n/space)
```

```
      DO I= 2, var5 + 1
         X2=SPACE*(DBLE(I-1)/DBLE(N))
         DMTMP(I)=DMP1/(COSH(DMP2*X2))**2.D0
         DM(I)=DMTMP(I)
         DM(N+2-I)=DM(I)
         DMTMP(N+2-I)=DMTMP(I)
      enddo

C..DAMPING LAYER

      var5=aint(NDP1*N/SPACE)
      var6=aint(NDP2*N/SPACE)
      DO I=var5+2, var6+1
         X=1.D0-((I-1)*SPACE/N-NDP1)/(NDP2-NDP1)
         F=(126.D0-420.D0*X+540.D0*X**2.D0-315.D0*X**3.D0
     &            +70.D0*X**4.D0)*X**5.D0
         X2=SPACE*(DBLE(I-1)/DBLE(N))
         DMTMP(I)=DMP1/(COSH(DMP2*X2))**2.D0
         DM(I)=F*DMTMP(I)
         DMTMP(N+2-I)=DMTMP(I)
         DM(N+2-I)=DM(I)
      enddo

C..INTERIOR LAYER

      var6=aint(ndp2*n/space)
      DO I= var6+2, N/2+1
         X2=SPACE*(DBLE(I-1)/DBLE(N))
         DMTMP(I)=DMP1/(COSH(DMP2*X2))**2.D0
         DM(I)=0.D0
         DMTMP(N+2-I)=DMTMP(I)
         DM(N+2-I)=DM(I)
      enddo

      IF (IDEBUG .EQ. 1) THEN

         DO I=1,N
            X2=SPACE*(DBLE(I-1)/DBLE(N))-SPACE/2.D0
         enddo

      ENDIF

C..INITIALIZING THE FOURIER TRANSFORM AND INTEGRATING FACTORS
      PI=ACOS(-1.D0)
      CI=(0.D0,1.D0)
      CALL FFTI(N)

C..FORWARD INTEGRATING FACTOR
      DO K=2,N/2+1
         AK=2.D0*PI/SPACE*DBLE(K-1)
         INTF(K)=EXP(CI*diff1*(AK**2.D0/2.D0)*DT/2.D0)
         INTB(K)=1.D0/INTF(K)
         INTF(N+2-K)=EXP(CI*diff1*(AK**2.D0/2.D0)*DT/2.D0)
         INTB(N+2-K)=1.D0/INTF(N+2-K)
         INTF2(K)=EXP(CI*diff2*(AK**2.D0/2.D0)*DT/2.D0)
         INTB2(K)=1.D0/INTF2(K)
         INTF2(N+2-K)=EXP(CI*diff2*(AK**2.D0/2.D0)*DT/2.D0)
         INTB2(N+2-K)=1.D0/INTF2(N+2-K)
      enddo
      INTF(1)=1.0D0
      INTB(1)=1.0D0
      INTF2(1)=1.0D0
      INTB2(1)=1.0D0

*
      END
*
**********************************************************************
*
      SUBROUTINE RUNGE_KUTTA
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      COMPLEX*16 UT,U0,U0T,NON,NONT,SOLNT
      COMPLEX*16 U2T,U20,U20T,NON2,NON2T,SOLN2T
      COMPLEX*16 TFF,TFFT,U21
```

```
      COMPLEX*16 TFF2,TFF2T,U22
      COMPLEX*16 CI,U,INTF,INTB,U2,INTF2,INTB2
      PARAMETER (MAX=2048,MAXY=1024)
      DIMENSION U0(MAX,MAXY),U0T(MAX,MAXY),TEMP(MAX,MAXY)
      DIMENSION U20(MAX,MAXY),U20T(MAX,MAXY)
      DIMENSION U21(MAX,MAXY),U22(MAX,MAXY)
      DIMENSION DM(MAX),TFF(MAX),TFFT(MAX)
      DIMENSION TFF2(MAX),TFF2T(MAX)
      DIMENSION NON(MAX,MAXY),NONT(MAX,MAXY),SOLNT(MAX,MAXY)
      DIMENSION NON2(MAX,MAXY),NON2T(MAX,MAXY),SOLN2T(MAX,MAXY)
      DIMENSION UT(MAX,MAXY)
      DIMENSION U2T(MAX,MAXY)

      DIMENSION U(MAX,MAXY),INTF(MAX),INTB(MAX)
      DIMENSION U2(MAX,MAXY),INTF2(MAX),INTB2(MAX)

      COMMON/SOLN/U,U0,U2,U20,TEMP
      COMMON/INTEGERS/N,NY,NSTEPS,IMID,IDEBUG,NSX,NSY
      COMMON/INT/INTF,INTB,INTF2,INTB2
      COMMON/TIME/DT,T0,TF,SRF
      COMMON/LENGTH/SPACE,SPACEY
      COMMON/AMP/A1,A2,W1,W2,R,EPS,OM,XD,YD,V,XD2,YD2,V2,
     &      xmin,xmax,ymin,ymax,TargetX
      COMMON/NEWPAR/rnu,th0,te00,te01,qs,diff1,diff2,AAAA,BBBB
      COMMON/DAMP/DM

C..FULL 4TH ORDER ACCURATE RUNGE-KUTTA SCHEME
C          U1 = +U0 + (DT/2) F(U0,0)
C          U2 = +U0 + (DT/2) F(U1,DT/2)
C          U3 = +U0 + (DT) F(U2,DT/2)
C          U4 = -U0 + (DT/2) F(U3,DT)
C
C          UF = (1/3)U1 + (2/3)U2 + (1/3)U3 + (1/3)U4
C
C... Caveat: this routine uses Runge-Kutta in the Fourier domain

C..PARAMETERS NEEDED FOR THIS ROUTINE
      CI=(0.D0,1.D0)
      PI=ACOS(-1.D0)

C..INITALIZING U0 AND TRANSFORMING TO THE SPECTRAL DOMAIN
      DO 11 K=1,NY
         DO 10 I=1,N
            U0(I,K)=U(I,K)
            U20(I,K)=U2(I,K)
  10     CONTINUE
  11  CONTINUE

      DO 641 K=1,NY
         DO 642 I=1,N
            TFF(I)=U0(I,K)
            TFF2(I)=U20(I,K)
 642     CONTINUE
         CALL FFTF(TFF,TFFT,N)
         CALL FFTF(TFF2,TFF2T,N)
         DO 643 I=1,N
            U0T(I,K)=TFFT(I)
            U20T(I,K)=TFF2T(I)
 643     CONTINUE
 641  CONTINUE

C.. Calculate second y derivative of U

      CALL FFTI(NY)
      DO 775 I=1,N
         DO 776 K=1,NY
            TFF(K)=U(I,K)
            TFF2(K)=U2(I,K)
 776     CONTINUE
         CALL FFTF(TFF,TFFT,NY)
         CALL FFTF(TFF2,TFF2T,NY)
         DO 910 K=2,NY/2+1
            AK=2.D0*PI/SPACEY*DBLE(K-1)
            TFFT(K)=-AK*AK*TFFT(K)/DBLE(NY)
```

```
            TFFT(NY+2-K)=-AK*AK*TFFT(NY+2-K)/DBLE(NY)
            TFF2T(K)=-AK*AK*TFF2T(K)/DBLE(NY)
            TFF2T(NY+2-K)=-AK*AK*TFF2T(NY+2-K)/DBLE(NY)
 910     CONTINUE
         TFFT(1)=0.0D0
         TFF2T(1)=0.0D0
         CALL FFTB(TFFT,TFF,NY)
         CALL FFTB(TFF2T,TFF2,NY)
         DO 777 K=1,NY
            U21(I,K)=TFF(K)
            U22(I,K)=TFF2(K)
 777     CONTINUE
 775  CONTINUE

      CALL FFTI(N)

C...1 CALCULATING THE NONLINEARITY AND TRANSFORMING TO SPECTRAL DOMAIN
C...  nonlinearity evaluated at t
C
      DO 121 K=1,NY
         DO 120 I=1,N
C
            tem=dsin(2.d0*temp(i,k))
            NON(I,K)=CI*AAAA*tem*U(I,K)+0.5d0*diff1*CI*U21(I,K)
     &       -DM(I)*U(I,K)
            NON2(I,K)=CI*BBBB*tem*U2(I,K)+0.5d0*diff2*CI*U22(I,K)
     &       -DM(I)*U2(I,K)
 120     CONTINUE

         DO 321 I=1,N
            TFF(I)=NON(I,K)
            TFF2(I)=NON2(I,K)
 321     CONTINUE
         CALL FFTF(TFF,TFFT,N)
         CALL FFTF(TFF2,TFF2T,N)
         DO 322 I=1,N
            NONT(I,K)=TFFT(I)
            NON2T(I,K)=TFF2T(I)
 322     CONTINUE

         DO 130 KK=1,N
C...1 FORWARD ZERO-PROPAGATE AND HALF-STEP
C...  nonlinearity evaluated at t so no extra exponential factor
C
            UT(KK,K)=U0T(KK,K)+DT/2.D0*NONT(KK,K)
            U2T(KK,K)=U20T(KK,K)+DT/2.D0*NON2T(KK,K)
C
C...1 UPDATE SOLUTION... in Fourier domain
C
            SOLNT(KK,K)=(1.D0/3.D0)*UT(KK,K)
            SOLN2T(KK,K)=(1.D0/3.D0)*U2T(KK,K)
C
C...2 BACKWARD HALF-PROPAGATE... at next step will need to calculate
C...  nonlinearity at t+dt/2, so need U at t+dt/2; use UT to get it
C
            UT(KK,K)=UT(KK,K)*INTB(KK)/DBLE(N)
            U2T(KK,K)=U2T(KK,K)*INTB2(KK)/DBLE(N)
 130     CONTINUE

         DO 323 I=1,N
            TFFT(I)=UT(I,K)
            TFF2T(I)=U2T(I,K)
 323     CONTINUE
         CALL FFTB(TFFT,TFF,N)
         CALL FFTB(TFF2T,TFF2,N)
         DO 324 I=1,N
            U(I,K)=TFF(I)
            U2(I,K)=TFF2(I)
 324     CONTINUE
 121  CONTINUE

C.. Calculate second y derivative of U

      CALL FFTI(NY)
```

```
      DO 785 I=1,N
         DO 786 K=1,NY
            TFF(K)=U(I,K)
            TFF2(K)=U2(I,K)
 786     CONTINUE
         CALL FFTF(TFF,TFFT,NY)
         CALL FFTF(TFF2,TFF2T,NY)
         DO 980 K=2,NY/2+1
            AK=2.D0*PI/SPACEY*DBLE(K-1)
            TFFT(K)=-AK*AK*TFFT(K)/DBLE(NY)
            TFFT(NY+2-K)=-AK*AK*TFFT(NY+2-K)/DBLE(NY)
            TFF2T(K)=-AK*AK*TFF2T(K)/DBLE(NY)
            TFF2T(NY+2-K)=-AK*AK*TFF2T(NY+2-K)/DBLE(NY)
 980     CONTINUE
         TFFT(1)=0.0D0
         TFF2T(1)=0.0D0
         CALL FFTB(TFFT,TFF,NY)
         CALL FFTB(TFF2T,TFF2,NY)
         DO 787 K=1,NY
            U21(I,K)=TFF(K)
            U22(I,K)=TFF2(K)
 787     CONTINUE
 785  CONTINUE

      CALL FFTI(N)

C...2 CALCULATING THE NONLINEARITY AND TRANSFORMING TO SPECTRAL DOMAIN
C...  evaluate nonlinearity at t+dt/2
C
      DO 221 K=1,NY
         DO 220 I=1,N
            tem=dsin(2.d0*temp(i,k))
            NON(I,K)=CI*AAAA*tem*U(I,K)
     &        +0.5d0*diff1*CI*U21(I,K)-DM(I)*U(I,K)
            NON2(I,K)=CI*BBBB*tem*U2(I,K)
     &         +0.5d0*diff2*CI*U22(I,K)-DM(I)*U2(I,K)
 220     CONTINUE

         DO 331 I=1,N
            TFF(I)=NON(I,K)
            TFF2(I)=NON2(I,K)
 331     CONTINUE
         CALL FFTF(TFF,TFFT,N)
         CALL FFTF(TFF2,TFF2T,N)
         DO 332 I=1,N
            NONT(I,K)=TFFT(I)
            NON2T(I,K)=TFF2T(I)
 332     CONTINUE


         DO 230 KK=1,N
C...2 FORWARD HALF-PROPAGATE AND HALF-STEP
C...  nonlinearity evaluated at t+dt/2 so one extra exponential factor
C
            UT(KK,K)=U0T(KK,K)+DT/2.D0*INTF(KK)*NONT(KK,K)
            U2T(KK,K)=U20T(KK,K)+DT/2.D0*INTF2(KK)
     &       *NON2T(KK,K)
C
C...2 UPDATE SOLUTION... in Fourier domain
C
            SOLNT(KK,K)=SOLNT(KK,K)+(2.D0/3.D0)*UT(KK,K)
            SOLN2T(KK,K)=SOLN2T(KK,K)+(2.D0/3.D0)*U2T(KK,K)
C
C...3 BACKWARD HALF-PROPAGATE... at next step will need to calculate
C...  nonlinearity at t+dt/2, so need U at t+dt/2; use UT to get it
C
            UT(KK,K)=UT(KK,K)*INTB(KK)/DBLE(N)
            U2T(KK,K)=U2T(KK,K)*INTB2(KK)/DBLE(N)
 230     CONTINUE
         DO 333 I=1,N
            TFFT(I)=UT(I,K)
            TFF2T(I)=U2T(I,K)
 333     CONTINUE
```

```
         CALL FFTB(TFFT,TFF,N)
         CALL FFTB(TFF2T,TFF2,N)
         DO 334 I=1,N
            U(I,K)=TFF(I)
            U2(I,K)=TFF2(I)
 334     CONTINUE
 221  CONTINUE

C.. Calculate second y derivative of U

      CALL FFTI(NY)

      DO 795 I=1,N
         DO 796 K=1,NY
            TFF(K)=U(I,K)
            TFF2(K)=U2(I,K)
 796     CONTINUE
         CALL FFTF(TFF,TFFT,NY)
         CALL FFTF(TFF2,TFF2T,NY)
         DO 990 K=2,NY/2+1
            AK=2.D0*PI/SPACEY*DBLE(K-1)
            TFFT(K)=-AK*AK*TFFT(K)/DBLE(NY)
            TFFT(NY+2-K)=-AK*AK*TFFT(NY+2-K)/DBLE(NY)
            TFF2T(K)=-AK*AK*TFF2T(K)/DBLE(NY)
            TFF2T(NY+2-K)=-AK*AK*TFF2T(NY+2-K)/DBLE(NY)
 990     CONTINUE
         TFFT(1)=0.0D0
         TFF2T(1)=0.0D0
         CALL FFTB(TFFT,TFF,NY)
         CALL FFTB(TFF2T,TFF2,NY)
         DO 797 K=1,NY
            U21(I,K)=TFF(K)
            U22(I,K)=TFF2(K)
 797     CONTINUE
 795  CONTINUE

      CALL FFTI(N)

C...3 CALCULATING THE NONLINEARITY AND TRANSFORMING TO SPECTRAL DOMAIN
C...  evaluate nonlinearity at t+dt/2
C
      DO 821 K=1,NY
         DO 820 I=1,N
            tem=dsin(2.d0*temp(i,k))
            NON(I,K)=CI*tem*AAAA*U(I,K)
     &       +0.5d0*diff1*CI*U21(I,K)-DM(I)*U(I,K)
            NON2(I,K)=CI*tem*BBBB*U2(I,K)
     &       +0.5d0*diff2*CI*U22(I,K)-DM(I)*U2(I,K)
 820     CONTINUE

         DO 341 I=1,N
           TFF(I)=NON(I,K)
           TFF2(I)=NON2(I,K)
 341     CONTINUE
         CALL FFTF(TFF,TFFT,N)
         CALL FFTF(TFF2,TFF2T,N)
         DO 342 I=1,N
            NONT(I,K)=TFFT(I)
            NON2T(I,K)=TFF2T(I)
 342     CONTINUE

         DO 330 KK=1,N
C...3 FORWARD HALF-PROPAGATE AND FULL-STEP
C...  nonlinearity evaluated at t+dt/2 so one extra exponential factor
C
            UT(KK,K)=UOT(KK,K)+DT*INTF(KK)*NONT(KK,K)
            U2T(KK,K)=U2OT(KK,K)
     &       +DT*INTF2(KK)*NON2T(KK,K)
C
C...3 UPDATE SOLUTION... in Fourier domain
C
            SOLNT(KK,K)=SOLNT(KK,K)+(1.DO/3.DO)*UT(KK,K)
            SOLN2T(KK,K)=SOLN2T(KK,K)+(1.DO/3.DO)*U2T(KK,K)
C
```

```
C...4 BACKWARD FULL-PROPAGATE... at next step will need to calculate
C...  nonlinearity at t+dt, so need U at t+dt; use UT to get it
C
            UT(KK,K)=UT(KK,K)*INTB(KK)*INTB(KK)/DBLE(N)
            U2T(KK,K)=U2T(KK,K)*INTB2(KK)*INTB2(KK)/DBLE(N)
 330     CONTINUE

         DO 343 I=1,N
            TFFT(I)=UT(I,K)
            TFF2T(I)=U2T(I,K)
 343     CONTINUE
         CALL FFTB(TFFT,TFF,N)
         CALL FFTB(TFF2T,TFF2,N)
         DO 344 I=1,N
            U(I,K)=TFF(I)
            U2(I,K)=TFF2(I)
 344     CONTINUE
 821  CONTINUE

C.. Calculate second y derivative of U

      CALL FFTI(NY)

      DO 715 I=1,N
         DO 716 K=1,NY
            TFF(K)=U(I,K)
            TFF2(K)=U2(I,K)
 716     CONTINUE
         CALL FFTF(TFF,TFFT,NY)
         CALL FFTF(TFF2,TFF2T,NY)
         DO 916 K=2,NY/2+1
            AK=2.D0*PI/SPACEY*DBLE(K-1)
            TFFT(K)=-AK*AK*TFFT(K)/DBLE(NY)
            TFFT(NY+2-K)=-AK*AK*TFFT(NY+2-K)/DBLE(NY)
            TFF2T(K)=-AK*AK*TFF2T(K)/DBLE(NY)
            TFF2T(NY+2-K)=-AK*AK*TFF2T(NY+2-K)/DBLE(NY)
 916     CONTINUE
         TFFT(1)=0.0D0
         TFF2T(1)=0.0D0
         CALL FFTB(TFFT,TFF,NY)
         CALL FFTB(TFF2T,TFF2,NY)
         DO 717 K=1,NY
            U21(I,K)=TFF(K)
            U22(I,K)=TFF2(K)
 717     CONTINUE
 715  CONTINUE

      CALL FFTI(N)

C...4 CALCULATING THE NONLINEARITY AND TRANSFORMING TO SPECTRAL DOMAIN
C...  evaluate nonlinearity at t+dt
C
      DO 421 K=1,NY
         DO 420 I=1,N
            tem=dsin(2.d0*temp(i,K))
            NON(I,K)=CI*AAAA*tem*U(I,K)
     &       +0.50d0*diff1*CI*U21(I,K)-DM(I)*U(I,K)
            NON2(I,K)=CI*BBBB*tem*U2(I,K)
     &       +0.50d0*diff2*CI*U22(I,K)-DM(I)*U2(I,K)
 420     CONTINUE

         DO 351 I=1,N
           TFF(I)=NON(I,K)
           TFF2(I)=NON2(I,K)
 351     CONTINUE
         CALL FFTF(TFF,TFFT,N)
         CALL FFTF(TFF2,TFF2T,N)
         DO 352 I=1,N
            NONT(I,K)=TFFT(I)
            NON2T(I,K)=TFF2T(I)
 352     CONTINUE

         DO 430 KK=1,N
C...4 FORWARD FULL-PROPAGATE AND HALF-STEP
```

```
C...  nonlinearity evaluated at t+dt so two extra exponential factors
C
            UT(KK,K)=-U0T(KK,K)+DT/2.D0*INTF(KK)*INTF(KK)*NONT(KK,K)
            U2T(KK,K)=-U20T(KK,K)
     &       +DT/2.D0*INTF2(KK)*INTF2(KK)*NON2T(KK,K)
C
C...4 FINAL SOLUTION AND BACKWARD FULL-PROPAGATE
C...  two exponential factors needed to convert back to original vars
C
            UT(KK,K)=(SOLNT(KK,K)+(1.D0/3.D0)*UT(KK,K))*INTB(KK)
     &             *INTB(KK)/DBLE(N)
            U2T(KK,K)=(SOLN2T(KK,K)+(1.D0/3.D0)*U2T(KK,K))*INTB2(KK)
     &             *INTB2(KK)/DBLE(N)
 430     CONTINUE

C...4 REVERSE FOURIER TRANSFORM TO PHYSICAL SPACE

         DO 353 I=1,N
            TFFT(I)=UT(I,K)
            TFF2T(I)=U2T(I,K)
 353     CONTINUE
         CALL FFTB(TFFT,TFF,N)
         CALL FFTB(TFF2T,TFF2,N)
         DO 354 I=1,N
            U(I,K)=TFF(I)
            U2(I,K)=TFF2(I)
 354     CONTINUE
 421  CONTINUE

      RETURN
      END
*
*****************************************************************
*
      SUBROUTINE HEAT
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      COMPLEX*16 U,U0,RHT,TT,RHY
      COMPLEX*16 U2,U20
      COMPLEX*16 TE,TEMB,TEMF,RH,RHE
      PARAMETER (MAX=2048,MAXY=1024)
      DIMENSION U(MAX,MAXY),TEMP(MAX,MAXY)
      DIMENSION U0(MAX,MAXY),RHT(MAX),RH(MAX)
      DIMENSION U2(MAX,MAXY),U20(MAX,MAXY)
      DIMENSION TT(MAX),RHE(MAX)
      DIMENSION DIS(MAX),RHY(MAX,MAXY),TE(MAX)
      DIMENSION TEMB(MAX),TEMF(MAX,MAXY)
      DIMENSION AAA(MAX),BBB(MAX),CCC(MAX)
      DOUBLE PRECISION NDP1,NDP2,Wrk7,Wrk8
      DOUBLE PRECISION Wrk0,Wrk1,Wrk2,Wrk3,Wrk4,Wrk5,Wrk6

      COMMON/INTEGERS/N,NY,NSTEPS,IMID,IDEBUG,NSX,NSY
      COMMON/TIME/DT,T0,TF,SRF
      COMMON/LENGTH/SPACE,SPACEY
      COMMON/AMP/A1,A2,W1,W2,R,EPS,OM,XD,YD,V,XD2,YD2,V2,
     &      xmin,xmax,ymin,ymax,TargetX
      COMMON/NEWPAR/rnu,th0,te00,te01,qs,diff1,diff2,AAAA,BBBB
      COMMON/CONTINUE/NCONT
      COMMON/ASORB/NDP1,NDP2,DMP1,DMP2
      COMMON/SOLN/U,U0,U2,U20,TEMP

      dx=SPACE/DFLOAT(N)
      dy=SPACEY/dble(NY)
      pi=dacos(-1.d0)

      jj=0

      CALL FFTI(N)

 777  continue

      jj=jj+1

      Wrk0=2.0D0
      Wrk5=-Wrk0*AAAA
```

```
      Wrk6=-Wrk0*BBBB
      do 11 J=1,NY
         do 10 I=1,N
            Wrk1=Wrk0*temp(i,j)
            Wrk2=abs(u(i,j))
            Wrk3=abs(u2(i,j))
            Wrk4=dcos(Wrk1)
            re1=qs*(dsin(Wrk1)-Wrk1)
            re2=Wrk5*Wrk4*Wrk2*Wrk2
            re3=Wrk6*Wrk4*Wrk3*Wrk3
            RHE(I)=dcmplx(re1+re2+re3,0.0d0)
10       CONTINUE

         CALL FFTF(RHE,RHT,N)

         do 12 I=1,N
            RHY(I,J)=RHT(I)
12       continue
11    continue

      do 13 I=2,N/2+1
        AK=2.D0*PI/SPACE*DBLE(I-1)
        DIS(I)=-AK*AK
        DIS(N+2-I)=-AK*AK
13    continue
      DIS(1)=0.0d0

      DO 35 I=1,N
         do 20 J=2,NY-1
            AAA(J-1)=1.0d0
            BBB(J-1)=-2.0d0*(1.0d0+qs*dy*dy/rnu)
     &           +dy*dy*DIS(I)
            CCC(J-1)=1.0d0
20       continue
         DO 30 J=2,NY-1
            rh(J-1)=dy*dy*RHY(I,J)/rnu
30       continue

         call tridag(AAA,BBB,CCC,rh,te,ny-2)

         do 40 J=1,NY-2
            TEMF(I,J+1)=te(J)/dble(N)
40       continue
         temf(I,1)=dcmplx(0.0d0,0.0d0)
         temf(I,NY)=dcmplx(0.0d0,0.0d0)
35    continue

      rmax=0.0d0
      DO 60 J=2,NY-1
         DO 65 I=1,N
            TT(I)=TEMF(I,J)
65       continue

         CALL FFTB(TT,TEMB,N)

         do 45 I=2,N-1
            Wrk7=temp(i,j)
            Wrk8=Dreal(temb(i))
            temp(i,j)=Wrk8
            rmax1=Dabs(Wrk8-Wrk7)
            if(rmax1.gt.rmax) rmax=rmax1
45       continue
60    continue
      do 70 I=1,N
         temp(I,1)=0.0d0
         temp(I,NY)=0.0d0
70    continue
      do 71 J=1,NY
         temp(1,J)=0.0d0
         temp(N,J)=0.0d0
71    continue

      if ((jj.gt.10).and.(Idebug.eq.1)) then
         write(6,*) 'HEAT rmax = ',rmax,' inter = ',jj
```

```
         call Wtime('Iter ')
      end if

      if(jj.gt.1000) then
         write(6,*) 'too many iterations', jj
         call Wtime('End ')
         stop
      end if
      if(rmax.gt.1.0d-7) go to 777

      return
      end

*
*********************************************************************
*
        SUBROUTINE FFTF(FOLD,F,N)
C  FAST FOURIER TRANSFORM (FORWARD) FROM NUMERICAL RECIPES
C  INPUT ARRAY F OLD WITH REAL AND IMAGINARY PARTS IN ALTERNATE CELLS
C  UPON RETURN F CONTAINS ITS FOURIER TRANSFORM
        INTEGER    NMAX
        PARAMETER (NMAX=2048)
        COMPLEX*16 FOLD(NMAX),F(NMAX),W
        REAL*8          CN(NMAX),SN(NMAX)
        COMMON/CFFTF/CN       ,SN
        COMPLEX*16      X(NMAX)
        COMMON/CFFT/ X

        DO 5 K=1,N
           F(K)=FOLD(K)
5       CONTINUE
        NS=1
        NR=2
        NQ=N
11      DO 10 K=NR,N
           IF(MOD(NQ,K).EQ.0)GOTO 21
10      CONTINUE
21      ND=NQ/K
        NS=NS*K
        NR=K
        IQ=0
        ID=0
        DO 22 I=1,NS
           DO 24 J=1,ND
              L=IQ+J
              LP=L+ND
              M=ID
              W=F(L)+F(LP)*DCMPLX(CN(M+1),SN(M+1))
              IF(NR.EQ.2)GOTO 100
              L=LP
              DO 26 K=3,NR
                 L=L+ND
                 M=M+ID
                 IF(M.GE.N)M=M-N
                 W=W+F(L)*DCMPLX(CN(M+1),SN(M+1))
26            Continue
100           X(ID+J)=W
24         Continue
           ID=ID+ND
           IQ=IQ+NQ
           IF(IQ.GE.N)IQ=IQ-N
22      CONTINUE
        NQ=ND
        IF(ND.GT.1)GOTO 61
        DO 32 K=1,N
           F(K)=X(K)
32      Continue
        RETURN
61      DO 60 K=NR,N
           IF(MOD(NQ,K).EQ.0)GOTO 71
60      CONTINUE
71      ND=NQ/K
        NS=NS*K
        NR=K
```

```
        IQ=0
        ID=0
        DO 72 I=1,NS
           DO 74 J=1,ND
              L=IQ+J
              LP=L+ND
              M=ID
              W=X(L)+X(LP)*DCMPLX(CN(M+1),SN(M+1))
              IF(NR.EQ.2)GOTO 110
              L=LP
              DO 76 K=3,NR
                 L=L+ND
                 M=M+ID
                 IF(M.GE.N)M=M-N
                 W=W+X(L)*DCMPLX(CN(M+1),SN(M+1))
76            Continue
110           F(ID+J)=W
74         Continue
           ID=ID+ND
           IQ=IQ+NQ
           IF(IQ.GE.N)IQ=IQ-N
72      CONTINUE
        NQ=ND
        IF(ND.GT.1)GOTO 11
        RETURN
        END
C
C=====================================================================*
C
        SUBROUTINE FFTB(FOLD,F,N)
C  FAST FOURIER TRANSFORM (REVERSE) FROM NUMERICAL RECIPES
C  INPUT ARRAY F OLD WITH REAL AND IMAGINARY PARTS IN ALTERNATE CELLS
C  UPON RETURN F CONTAINS ITS FOURIER TRANSFORM
        INTEGER   NMAX
        PARAMETER (NMAX=2048)
        COMPLEX*16 FOLD(NMAX),F(NMAX),W
        REAL*8         CN(NMAX),SN(NMAX)
        COMMON/CFFTB/CN      ,SN
        COMPLEX*16     X(NMAX)
        COMMON/CFFT/X
C
        DO 5 K=1,N
           F(K)=FOLD(K)
5       CONTINUE
        NS=1
        NR=2
        NQ=N
11      DO 10 K=NR,N
           IF(MOD(NQ,K).EQ.0)GOTO 21
10      CONTINUE
21      ND=NQ/K
        NS=NS*K
        NR=K
        IQ=0
        ID=0
        DO 22 I=1,NS
           DO 24 J=1,ND
              L=IQ+J
              LP=L+ND
              M=ID
              W=F(L)+F(LP)*DCMPLX(CN(M+1),SN(M+1))
              IF(NR.EQ.2)GOTO 100
              L=LP
              DO 26 K=3,NR
                 L=L+ND
                 M=M+ID
                 IF(M.GE.N)M=M-N
                 W=W+F(L)*DCMPLX(CN(M+1),SN(M+1))
26            Continue
100           X(ID+J)=W
24         Continue
           ID=ID+ND
           IQ=IQ+NQ
           IF(IQ.GE.N)IQ=IQ-N
```

```
22      CONTINUE
        NQ=ND
        IF(ND.GT.1)GOTO 61
        DO 32 K=1,N
           F(K)=X(K)
32      Continue
        RETURN
61      DO 60 K=NR,N
           IF(MOD(NQ,K).EQ.0)GOTO 71
60      CONTINUE
71      ND=NQ/K
        NS=NS*K
        NR=K
        IQ=0
        ID=0
        DO 72 I=1,NS
           DO 74 J=1,ND
              L=IQ+J
              LP=L+ND
              M=ID
              W=X(L)+X(LP)*DCMPLX(CN(M+1),SN(M+1))
              IF(NR.EQ.2)GOTO 110
              L=LP
              DO 76 K=3,NR
                 L=L+ND
                 M=M+ID
                 IF(M.GE.N)M=M-N
                 W=W+X(L)*DCMPLX(CN(M+1),SN(M+1))
76            Continue
110           F(ID+J)=W
74         Continue
           ID=ID+ND
           IQ=IQ+NQ
           IF(IQ.GE.N)IQ=IQ-N
72      CONTINUE
        NQ=ND
        IF(ND.GT.1)GOTO 11
        RETURN
        END
C
C=====================================================================*
C
        SUBROUTINE FFTI(N)
C FAST FOURIER TRANSFORM INITIALIZATION - FROM NUMERICAL RECIPES
C INITIALIZES SINE AND COSINE VECTORS FOR FFT ROUTINES
C       IMPLICIT UNDEFINED(A-Z)
        INTEGER   NMAX
        PARAMETER (NMAX=2048)
        INTEGER N,J
        DOUBLE PRECISION PI2,DT,ANG,Cwrk,Swrk

        REAL*8         CNF(NMAX),SNF(NMAX)
        COMMON/CFFTF/CNF       ,SNF
        REAL*8         CNB(NMAX),SNB(NMAX)
        COMMON/CFFTB/CNB       ,SNB
        COMPLEX*16     X(NMAX)
        COMMON/CFFT/X

        PI2=6.28318530717958647700D0
        DT=(PI2)/DBLE(N)
        ANG=0.D0
        DO 5 J=1,N
           Cwrk=Dcos(Ang)
           Swrk=Dsin(Ang)
           CNB(J)=Cwrk
           SNB(J)=Swrk
           CNF(J)=Cwrk
           SNF(J)=-Swrk
           ANG=ANG+DT
5       continue
        RETURN
        END
C
C=====================================================================*
```

```
C
      SUBROUTINE TRIDAG(A,B,C,R,UU,N)
      PARAMETER (NMAX=4096)
      IMPLICIT DOUBLE PRECISION (a-h,o-z)
      COMPLEX*16 R,UU
      DIMENSION GAM(NMAX),A(N),B(N),C(N),R(N),UU(N)
      Integer J,N
!
!     A tridiagonal matrix. Use Gaussian elimination to get the
!     solution. The input in not in the form of a matrix but in
!     the form of three vectors.
!
      BET=B(1)
      If (Bet.eq.0) goto 100
      UU(1)=R(1)/BET
      DO J=2,N
        GAM(J)=C(J-1)/BET
        BET=B(J)-A(J)*GAM(J)
        IF(BET.EQ.0.) goto 100
        UU(J)=(R(J)-A(J)*UU(J-1))/BET
      Enddo
      DO J=N-1,1,-1
        UU(J)=UU(J)-GAM(J+1)*UU(J+1)
      Enddo
100   If (Bet.EQ.0) then
          Write (6,900) J, Bet
          Stop
      Endif
900   Format ('Attempted division by zero ',I4,F28.16)
      END
C
C=======================================================================*
C
      subroutine maxfind(aa,max,pos,h,nn)
c****
c**    This subroutine finds the maximum element of the vector a of
c**    length num using a cubic spline interpolation.
c****
c
       IMPLICIT DOUBLE PRECISION (A-H,O-Z)
       parameter(nmax=2048)
       DOUBLE PRECISION aa(nmax),max,pos,h,deriv,d
       DOUBLE PRECISION xp(nmax),cp(4,nmax)
       integer begin,end,nn,num2,ipos,jpos
       double precision R,EPS,OM
       COMMON/AMP/A1,A2,W1,W2,R,EPS,OM,XD,YD,V,XD2,YD2,V2,
     &      xmin,xmax,ymin,ymax,TargetX

       max=aa(1)
       pos=0.0
       do 53 i=1,nn
          if(aa(i).gt.max) then
             max=aa(i)
             pos=(i-1)*h
             ipos=i
          end if
  53   continue

       num2 = nn
       begin = ipos-nn/2
       end = ipos+nn/2-1
       jpos = nn/2 + 1

       do 54 j=1,nn
          k=j+ipos-1-nn/2
          i=k-int((k-1)/nn)*nn
          if (i.le.0) i=i+nn
          xp(j)=(k-1)*h
          cp(1,j)=aa(i)
  54   continue

       call cubspl(xp,cp,num2,0,0)

       deriv=ppvalu(xp,cp,num2-1,4,xp(jpos),1)
```

```
       if (deriv .lt. 0) jpos=jpos-1

       if ( abs(cp(4,jpos)) .lt. 1.0E-4 ) then
           t = -cp(2,jpos)/cp(3,jpos)
       else
           d=sqrt(cp(3,jpos)*cp(3,jpos)-2.0*cp(2,jpos)*cp(4,jpos))
           t=(-cp(3,jpos)+d)/cp(4,jpos)
           if ((t.lt.0.) .OR. (t.gt.h)) t=t-2.0*d/cp(4,jpos)
       endif
       max=ppvalu(xp,cp,num2-1,4,xp(jpos)+t,0)
       pos=xp(jpos)+t

       return
       end
*
**********************************************************************
*
      Subroutine Wtime(A)
!
!     This subroutine prints out the time with the message A. So
!     calling Wtime('Start') will print out 'Start' and the time.
!
      INTEGER*4 Now(3)
      Character (*) A

      call itime(now)
      write(*,1000) A, ' ,time=', now(1), now(2), now(3)

1000  format(A20,A7, i2.2, ':', i2.2, ':', i2.2)
      End subroutine
*
**********************************************************************
*
      Double Precision Function PrimRad(Radians)
!
!     Convert the input value to range -Pi/2, Pi/2
!     Input is an angle but this is periodic so
!     convert the angle to the range -Pi/2 to Pi/2.
!
      Double precision  Radians, Pi

      Pi=3.1415926535897932385
      PrimRad=Radians-Pi/2.0D00 *int(Radians/Pi*2.0D00)
      Return
      End
*
**********************************************************************
*
```

The program to predict the next input velocity is shown below. This program is used in all programs where iteration is used to find the angle to direct he signal beam to the target value.

```
!*
!**********************************************************************
!*
real(kind=10) Function Predict(Xpos,Vel,iter,TarX,Xpos2,M_cross,M_Bnd)

implicit none
!
!     Predict Vel from Xpos using Lagrangian interpolation
!     An array is formed from Xpos and the Velocity (the angle
!     the laser beam is initially directed). The process is stopped
!     when Xpos is equal to the Target value of X.

integer, parameter :: Dbl=selected_real_kind(16,308)

real(Dbl), parameter    :: pi=3.141592654_Dbl

type experiment_setup
real(Dbl)   :: U, V
real(Dbl)   :: q, nu, D_u, D_v, step_size, start_x, step_x
real(Dbl)   :: Init_A_u, Init_B_u, Init_w_u, Init_xi_u
real(Dbl)   :: Init_A_v, Init_B_v, Init_w_v, Init_xi_v
```

```
real(Dbl)   :: Init_alpha_u, Init_beta_u
real(Dbl)   :: Init_alpha_v, Init_beta_v
integer     :: final_step, x_div, x_write
end type experiment_setup

type dark_sol_vect
real(Dbl)   :: A_u, B_u, w_u, beta_u, alpha_u, xi_u
real(Dbl)   :: A_v, B_v, w_v, beta_v, alpha_v, xi_v
real(Dbl)   :: z, u_min, v_min, xi_u_min, xi_v_min
end type dark_sol_vect

!
Type LS_iter
real(Dbl)   :: LS_Xpos
real(Dbl)   :: LS_Vel
real(Dbl)   :: LS_Xpos2
real(Dbl)   :: LS_M_cross
real(Dbl)   :: LS_M_Bnd
real(Dbl)   :: LS_W
real(Dbl)   :: LS_A_W
real(Dbl)   :: LS_A_Vel_W
real(Dbl)   :: LS_A_Vel_sqr_W
real(Dbl)   :: LS_Vel_W
real(Dbl)   :: LS_Vel_sqr_W
real(Dbl)   :: LS_Vel_cub_W
real(Dbl)   :: LS_Vel_for_W
End Type LS_iter

Type LS_table
type(LS_iter), dimension(25) :: LS
real(Dbl)   :: LS_TarX
real(Dbl)   :: LS_sum_w
real(Dbl)   :: LS_sum_A_w
real(Dbl)   :: LS_sum_A_Vel_w
real(Dbl)   :: LS_sum_A_Vel_sqr_w
real(Dbl)   :: LS_sum_Vel_w
real(Dbl)   :: LS_sum_Vel_sqr_w
real(Dbl)   :: LS_sum_Vel_cub_w
real(Dbl)   :: LS_sum_Vel_for_w
End Type LS_table

Type(LS_table) :: LS_tab

real(Dbl) Xpos,Xpos2,Vel,TarX
real(Dbl) Max_cross,Min_Bnd
real(Dbl) Wrk1,Wrk2,Wrk3
Integer iter, j, M_i, M_Cross, M_Bnd, k
Save LS_tab,M_i,Max_cross,Min_Bnd

Real(Dbl), dimension(3,4)   :: Gauss_elim
Real(Dbl)                   :: m, sum_v, Max_row, abs_ge
Real(Dbl)                   :: a, b, c, T, Vel_last
Integer                     :: i, l, copy_row_no, error_code
Real(Dbl), dimension(4)     :: Copy_row
Real(Dbl), dimension(3)     :: Approx_vect

! Write(6,*) 'Start Predict'

If (iter .eq. 1) then
M_i=0
Max_cross=  0.3D0
Min_bnd  = -1.5D0
Do j=1,25
LS_tab%LS(j)%LS_Xpos = 0.0_Dbl
LS_tab%LS(j)%LS_Vel = 0.0_Dbl
LS_tab%LS(j)%LS_Xpos2 = 0.0_Dbl
LS_tab%LS(j)%LS_M_cross = 0.0_Dbl
LS_tab%LS(j)%LS_M_Bnd = 0.0_Dbl
LS_tab%LS(j)%LS_w = 0.0_Dbl
LS_tab%LS(j)%LS_A_w = 0.0_Dbl
LS_tab%LS(j)%LS_A_Vel_w = 0.0_Dbl
LS_tab%LS(j)%LS_A_Vel_sqr_w = 0.0_Dbl
LS_tab%LS(j)%LS_Vel_w = 0.0_Dbl
LS_tab%LS(j)%LS_Vel_sqr_w = 0.0_Dbl
```

```
LS_tab%LS(j)%LS_Vel_cub_w = 0.0_Dbl
LS_tab%LS(j)%LS_Vel_for_w = 0.0_Dbl
End Do
LS_tab%LS_TarX = TarX
LS_tab%LS_sum_w = 0.0_Dbl
LS_tab%LS_sum_A_w = 0.0_Dbl
LS_tab%LS_sum_A_Vel_w = 0.0_Dbl
LS_tab%LS_sum_A_Vel_sqr_w = 0.0_Dbl
LS_tab%LS_sum_Vel_w = 0.0_Dbl
LS_tab%LS_sum_Vel_sqr_w = 0.0_Dbl
LS_tab%LS_sum_Vel_cub_w = 0.0_Dbl
LS_tab%LS_sum_Vel_for_w = 0.0_Dbl
Endif

LS_tab%LS(iter)%LS_Xpos = Xpos
LS_tab%LS(iter)%LS_Vel = Vel
LS_tab%LS(iter)%LS_Xpos2 = Xpos2
LS_tab%LS(iter)%LS_M_cross = M_Cross
LS_tab%LS(iter)%LS_M_Bnd = M_Bnd
Wrk1 = abs(Xpos-TarX)
LS_tab%LS(iter)%LS_w = Wrk1
LS_tab%LS(iter)%LS_A_w = Xpos*Wrk1
LS_tab%LS(iter)%LS_A_Vel_w = Xpos*Wrk1*Vel
LS_tab%LS(iter)%LS_A_Vel_sqr_w = Xpos*Wrk1*Vel*Vel
LS_tab%LS(iter)%LS_Vel_w = Wrk1*Vel
LS_tab%LS(iter)%LS_Vel_sqr_w = Wrk1*Vel*Vel
LS_tab%LS(iter)%LS_Vel_cub_w = Wrk1*Vel*Vel*Vel
LS_tab%LS(iter)%LS_Vel_for_w = Wrk1*Vel*Vel*Vel*Vel

LS_tab%LS_sum_w = Wrk1 + LS_tab%LS_sum_w
LS_tab%LS_sum_A_w = Xpos*Wrk1 + LS_tab%LS_sum_A_w
LS_tab%LS_sum_A_Vel_w = Xpos*Wrk1*Vel + LS_tab%LS_sum_A_Vel_w
LS_tab%LS_sum_A_Vel_sqr_w = Xpos*Wrk1*Vel*Vel + LS_tab%LS_sum_A_Vel_sqr_w
LS_tab%LS_sum_Vel_w = Wrk1*Vel + LS_tab%LS_sum_Vel_w
LS_tab%LS_sum_Vel_sqr_w = Wrk1*Vel*Vel + LS_tab%LS_sum_Vel_sqr_w
LS_tab%LS_sum_Vel_cub_w = Wrk1*Vel*Vel*Vel + LS_tab%LS_sum_Vel_cub_w
LS_tab%LS_sum_Vel_for_w = Wrk1*Vel*Vel*Vel*Vel + LS_tab%LS_sum_Vel_for_w

If (M_cross .eq. 1) then
If (Vel .lt. Max_cross) then
Max_cross =Vel
Endif
if (Xpos .lt. Xpos2) then
Wrk1=Vel+(Xpos2-Xpos)*1.5d-3
else
Wrk1=Vel+1.0d-3
Endif
ElseIf (M_bnd .eq. 1) then
If (Vel .gt. Min_Bnd) then
Min_Bnd=Vel
Endif
Wrk1=0.5d0*(Max_Cross+Min_Bnd)
Endif

! Write(6,*) ' Max Cross ',Max_Cross,' Min Bnd ',Min_Bnd,' Vel ', Vel
! Write(6,*) ' M_i ', M_i,'TarX',TarX,'Xpos',Xpos

if (iter .eq. 1) then
vel = LS_tab%LS(iter)%LS_Vel- 0.03_Dbl*(LS_tab%LS_TarX-LS_tab%LS(1)%LS_Xpos)
else if (iter .eq. 2) then
Wrk2 = (LS_tab%LS(2)%LS_Xpos-LS_tab%LS(1)%LS_Xpos)/(LS_tab%LS(2)%LS_Vel-LS_tab%LS(1)%LS_Vel)
else
Gauss_elim(1,1) = LS_tab%LS_sum_w
Gauss_elim(1,2) = LS_tab%LS_sum_Vel_w
Gauss_elim(1,3) = LS_tab%LS_sum_Vel_sqr_w
Gauss_elim(1,4) = LS_tab%LS_sum_A_w
Gauss_elim(2,1) = LS_tab%LS_sum_Vel_w
Gauss_elim(2,2) = LS_tab%LS_sum_Vel_sqr_w
Gauss_elim(2,3) = LS_tab%LS_sum_Vel_cub_w
Gauss_elim(2,4) = LS_tab%LS_sum_A_Vel_w
Gauss_elim(3,1) = LS_tab%LS_sum_Vel_sqr_w
Gauss_elim(3,2) = LS_tab%LS_sum_Vel_cub_w
Gauss_elim(3,3) = LS_tab%LS_sum_Vel_for_w
Gauss_elim(3,4) = LS_tab%LS_sum_A_Vel_sqr_w
```

```
do i=1,2
Max_row = abs(Gauss_elim(i,i))
copy_row_no = i
do l=i+1,3
abs_ge = abs(Gauss_elim(l,i))
if (abs_ge .GT. Max_row) then
Max_row = abs_ge
copy_row_no = l
end if
end do
if (Max_row .eq. 0.0_Dbl) then
error_code = 4
go to 100
end if
if (i .ne. copy_row_no) then
Copy_row = Gauss_elim(i,:)
Gauss_elim(i,:) = Gauss_elim(copy_row_no,:)
Gauss_elim(copy_row_no,:) = Copy_row
end if
do j=i+1,3
m = Gauss_elim(j,i)/Gauss_elim(i,i)
Gauss_elim(j,i)=0.0_Dbl
do k=i+1,4
Gauss_elim(j,k) = Gauss_elim(j,k)-m*Gauss_elim(i,k)
end do
end do
end do

Approx_vect(3) = Gauss_elim(3,4)/Gauss_elim(3,3)
do i=2,1,-1
sum_v=0.0_Dbl
do j=i+1,3
sum_v=sum_v+Gauss_elim(i,j)*Approx_vect(j)
end do
Approx_vect(i) = (Gauss_elim(i,4)-sum_v)/Gauss_elim(i,i)
end do
end if

100     continue
a = Approx_vect(1)
b = Approx_vect(2)
c = Approx_vect(3)
T = LS_tab%LS_TarX
Vel_last = LS_tab%LS(iter)%LS_Vel
Wrk1 = (-b+sqrt(b*b-4.0_Dbl*c*(a-T)))/(2.0_Dbl*c)
Wrk2 = (-b-sqrt(b*b-4.0_Dbl*c*(a-T)))/(2.0_Dbl*c)

If (abs(Wrk1-Vel_last).lt. abs(Wrk2-Vel_last)) then
Wrk3 = Wrk1
else
Wrk3 = Wrk2
End If
Predict=Wrk3

! Write(6,*) 'End Predict'

Return
End function Predict
```

The next program is used to calculate the results of the modulation equations.

```
Program Mod_tcol
parameter (nmax=16)
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
double precision y(nmax),dydx(nmax),h
double precision eta1, eta2, w1, w2, vx1, vx2, xi1, xi2
double precision e1, e2, d1, d2, AA, BB, tar_x
double precision eta1_0, eta2_0, w1_0, w2_0
double precision vx1_0, vx2_0, xi1_0, xi2_0, vx2_s
double precision e1_0, e2_0, d1_0, d2_0, AA_0, W_adj_0
double precision BB_0, tar_x_0,h_0,beti_s, arg, arg1
```

```
      double precision t,g1,r0,fa,W_max_grad,W_max_V,W_Min_V
      double precision rval(500000),rintv(2)
      double precision rval2(500000),rintv2(2)
      real out(nmax+1)
      integer*4 j,iflag,count,num,nn,mike,jj,iflag1, iter
      character*20 name
      character*4 yn
c
      common/param/eta100,w100,eta200,w200,r10,r20,h,mike
      common/flgblk/rval,rintv,rval2,rintv2,fa,iflag,jj,iflag1
      common/dblk/rnu,qs,d1,d2,AA,BB,cl1,cl2,cl10,cl20,W_adj,iflagr
      common/betblk/bet100,bet200
      common/momblk/xip
      common/guessblk/beti
c
      pi=dacos(-1.d0)
c
c Enter initial values
c
      RI=dlog(2.0d0)/3.0d0+1.0d0/6.0d0
      RI1=2.0d0*0.915965594d0
      RI2=dlog(2.0d0)
      RI4=2.0d0*dlog(2.0d0)/3.0d0-1.0d0/6.0d0
      RI42=2.0d0*dlog(2.0d0)/15.0d0+1.0d0/60.0d0
      RI7=1.352314016d0
      RI8=16.0d0*dlog(2.0d0)/35.0d0-19.0d0/105.0d0
c
      as=dsqrt(2.0d0)*RI2/dsqrt(RI7)
      bs=dsqrt(2.0d0*RI2)
      iter=1

      OPEN (15,FILE='modeqns.in',STATUS='OLD')
c      REWIND (15)

c..   change in time (delta t)
      read(15,220) h_0
c..   eta1 = initial amplitude for u
      READ (15,220) eta1_0
c..   eta2 = initial amplitude for v
      READ (15,220) eta2_0
c..   w1 = inital width for u
      READ (15,220) w1_0
c..   w2 = inital width for v
      READ (15,220) w2_0
c..   xi1 = x position for u
      READ (15,220) xi1_0
c..   e1 = y offset position for u
      READ (15,220) e1_0
c..   vx1 = velocity for u
      READ (15,220) vx1_0
c..   xi2 = x position for v
      READ (15,220) xi2_0
c..   e2 = y offset position for v
      READ (15,220) e2_0
c..   vx2 = velocity for v
      READ (15,220) vx2_0
c..   vy1 = velocity for u
      READ (15,220) vy1_0
c..   vy2 = velocity for v
      READ (15,220) vy2_0
c..   diff1 = diffraction coefficient for u
      READ (15,220) d1_0
c..   diff2 = diffraction coefficient for v
      READ (15,220) d2_0
c.    AA = A for u
      READ (15,220) AA_0
c.    BB = B for u
      READ (15,220) BB_0
c..   rnu = nu
      READ (15,220) rnu_0
c..   qs = q_{s}
      READ (15,220) qs_0
c..   num = plot interval
      READ (15,280) num
```

```
c..   tmax = maximum time
      READ (15,220) tmax_0
c..   mike = parameter for singular matrix
      READ (15,280) mike
c..   beti = initial guess for beta
      READ (15,220) beti_0
      beti_s=beti_0
      bet100=beti_0
c..   Target X position. Want to get a solution that ends at this X position
      READ (15,220) tar_x_0
c..   Width adjustment factor. If no width adjustment is 0.0
      READ (15,220) W_adj_0
      CLOSE (15)

        h = h_0
        eta1 = eta1_0
eta2 = eta2_0
w1 = w1_0
w2 = w2_0
xi1 = xi1_0
e1 = e1_0
vx1 = vx1_0
vy1 = vy1_0
xi2 = xi2_0
e2 = e2_0
vx2 = vx2_0
        vx2_s   = vx2_0
vy2 = vy2_0
d1 = d1_0
AA = AA_0
d2 = d2_0
BB = BB_0
rnu = rnu_0
qs = qs_0
        tmax = tmax_0
        beti = beti_s
        tar_x = tar_x_0
        W_adj= W_adj_0
        bet100 = beti

! W_max_grad=(Tar_x_0-xi2)/tmax
! W_max_V=datan(W_max_grad)
W_Max_V=0.78D0
W_Min_V=-0.78D0

c      IF (IDEBUG .EQ. 1) THEN
        WRITE (6,*)
        WRITE (6,260) h, '... change in the time step'
WRITE (6,260) eta1, '... initial amplitude for u'
WRITE (6,260) eta2, '... initial amplitude for v'
WRITE (6,260) w1, '... initial width for u'
WRITE (6,260) w2, '... initial width for v'
WRITE (6,260) xi1, '... x centre position for u'
WRITE (6,260) e1, '... y centre position for u'
WRITE (6,260) vx1, '... x velocity for u'
WRITE (6,260) vy1, '... y velocity for u'
WRITE (6,260) xi2, '... x centre position for v'
WRITE (6,260) e2, '... y centre position for v'
WRITE (6,260) vx2, '... x velocity for v'
WRITE (6,260) vy2, '... y velocity for v'
WRITE (6,260) d1,'... diffraction coefficient for u'
WRITE (6,260) AA,'A for u'
WRITE (6,260) d2,'... diffraction coefficient for v'
WRITE (6,260) BB,'B for v'
WRITE (6,260) rnu, '... nu'
WRITE (6,260) qs, '... q_{s}'
        WRITE (6,260) tmax, '... write output every t time steps'
        WRITE (6,240) num, '    ... plot interval'
        WRITE (6,240) mike, '    ... parameter for singular matrix'
        WRITE (6,260) beti, '... initial guess for beta'
        Write (6,260) tar_x, '...target value of X'
        Write (6,260) W_adj, '...width adjustment'
        Write (6,260) W_max_grad, '...Maximum gradient'
        Write (6,260) W_max_V, '...Maximum velocity'
```

```
 200  FORMAT (I10,14X,I6)
 220  FORMAT (F20.16)
 240  FORMAT (8X,I10,16X,A40)
 260  FORMAT (8X,F24.16,2X,A40)
 280  FORMAT (I6)


!      write(6,*) 'Input params 0 = ',eta1,w1,xi1
!      write(6,*) e1,w2,xi2,e2

      g1=0.0d0
      g2=0.0d0
      iflagr=0

      r10=eta1*eta1*w1*w1*RI2-eta100*eta100*w100*w100*RI2
      r10=r10/cl1
      r20=eta2*eta2*w2*w2*RI2-eta200*eta200*w200*w200*RI2
      r20=r20/cl2
      if(r10.lt.0.0d0) write(6,*) 'r10 < 0'
      if(r20.lt.0.0d0) write(6,*) 'r20 < 0'
      r10=dsqrt(dabs(r10))
      r20=dsqrt(dabs(r20))
c
      rmom00=RI2*eta1*eta1*w1*w1*vx1+RI2*eta2*eta2*w2*w2*vx2
      rmomy00=RI2*eta1*eta1*w1*w1*vy1+RI2*eta2*eta2*w2*w2*vy2
c
c      write(6,*) 'Enter name of datafile for amplitude'
c      read(5,*) name
c
c  Calculate entries of matrix for d.e.'s in initial form.
c

20    continue

 223  OPEN(UNIT=4,FILE='amp.dat')
      CLOSE(4,STATUS='DELETE')
      OPEN(UNIT=4,FILE='amp.dat',STATUS='NEW')
      REWIND(4)
      OPEN(UNIT=55,FILE='alpbet.dat')
      CLOSE(55,STATUS='DELETE')
      OPEN(UNIT=55,FILE='alpbet.dat',STATUS='NEW')
      REWIND(55)
      pi=dacos(-1.d0)
c
c     Reinitialise the variables
c

        h = h_0
        eta1 = eta1_0
eta2 = eta2_0
w1 = w1_0
w2 = w2_0
xi1 = xi1_0
e1 = e1_0
vx1 = vx1_0
vy1 = vy1_0
xi2 = xi2_0
e2 = e2_0
vx2 = vx2_s ! Need to use new value of vx2
vy2 = vy2_0
d1 = d1_0
AA = AA_0
d2 = d2_0
BB = BB_0
rnu = rnu_0
qs = qs_0
        tmax = tmax_0
        tar_x = tar_x_0
        W_adj= W_adj_0

      y(1)=eta1_0
      y(2)=w1_0
      y(3)=0.0d0
```

```
      y(4)=0.0d0
      y(5)=vx1_0
      y(6)=vy1_0
      y(7)=xi1_0
      y(8)=e1_0
      y(9)=eta2_0
      y(10)=w2_0
      y(11)=0.0d0
      y(12)=0.0d0
      y(13)=vx2_s ! Need to use new value of vx2
      y(14)=vy2_0
      y(15)=xi2_0
      y(16)=e2_0
      beti=beti_s ! Use new value of beta
c
c      write(6,*) 'Input params 1a = ',eta1,w1,xi1
c      write(6,*) e1,w2,xi2,e2
c      write(6,*) 'Input params 1b = ',y(1),y(2),y(7)
c      write(6,*) y(8),y(10),y(15),y(16)

c      write(6,*) 'Betafind t ',t

      call betafind(eta1,w1,xi1,e1,eta2,w2,xi2,e2,bet1,bet2)
      call afind(eta1,w1,xi1,e1,eta2,w2,xi2,e2,bet1,bet2,alpha1,alpha2)
c      write(6,*) 'beta1=',bet1
c      write(6,*) 'beta2=',bet2
c
c      write(6,*) 'alpha1=',alpha1
c      write(6,*) 'alpha2=',alpha2
c
      arg=(xi1-xi2)**2.0d0+(e1-e2)**2.0d0
      arg1=Width_calc(xi1,xi2,e1,e2,W_adj,w1,w2)
c      If (abs(arg-arg1) .gt. 1.0d-4) then
c      write(6,*) 'Main 1 arg1=',arg1,' arg=',arg,' W=',W_adj,w1,w2
c      endif
      earg1=dexp(-arg/(as*as*bet1*bet1+bs*bs*w1*w1))
      earg2=dexp(-arg/(2.0d0*as*as*bet1*bet1))
      te1=2.0d0*d1*RI*eta1*eta1
      te2=4.0d0*AA*as*as*bs*bs*alpha1*eta1*eta1*w1*w1*bet1*bet1
      te2=te2/(as*as*bet1*bet1+bs*bs*w1*w1)
      te3=te2*earg1
      te4=8.0d0*rnu*RI42*alpha1*alpha1
      te5=4.0d0*qs*RI4*alpha1*alpha1*bet1*bet1
      te6=rnu*alpha1*alpha1*(1.0d0-arg/(2.0d0*as*as*bet1*bet1))
      te6=te6*earg2
      te7=qs*as*as*alpha1*alpha1*bet1*bet1*earg2
      h0=te1-te2-te3+te4+te5+te6+te7
!      write(6,*) 'h0=',h0
c
      call fixedfind(h0,eta00,w00,bet00)
!      write(6,*) 'eta00=',eta00
!      write(6,*) 'w00=',w00
!      write(6,*) 'bet100=',bet00
c

!      write(6,*) 'Input params 1c = ',eta1,w1,xi1
!      write(6,*) e1,w2,xi2,e2

      bet100=bet00
      bet200=bet100
      w100=w00
      w200=w00
      re1=d1*RI*qs*(RI4+0.25d0*as*as)

c      write(6,*) 'd1 RI qs RI4 as = ',d1,RI,qs,RI4,as,re1

      re1=re1*(as*as*bet100*bet100+bs*bs*w100*w100)**4.0d0

c      write(6,*) 'as bet100 bs w100 = ',as,BET100,bs,w100

      re2=8.0d0*AA*AA*(as**4.0d0)*(bs**8.0d0)*(w100**8.0d0)
     &     *bet100**2.0d0

c      write(6,*) 'AA = ',AA
```

```
      eta100=re1/re2

c      write(6,*) 'Eta100 re1, re2 = ',re1,re2

      eta100=dsqrt(eta100)
      re1=d2*RI*qs*(RI4+0.25d0*as*as)
      re1=re1*(as*as*bet200*bet200+bs*bs*w200*w200)**4.0d0
      re2=8.0d0*BB*BB*(as**4.0d0)*(bs**8.0d0)*(w200**8.0d0)
     &     *bet200**2.0d0

c      write(6,*) 'Input params 1d = ',eta1,w1,xi1
c      write(6,*) e1,w2,xi2,e2

      eta200=re1/re2
      eta200=dsqrt(eta200)

c      write(6,*) 'Eta200 re1, re2 = ',re1,re2

c      write(6,*) 'eta100=',eta100
c      write(6,*) 'eta200=',eta200
      re1=2.0d0*as*RI1*RI1*dsqrt(rnu*RI42)*w100
      re2=3.0d0*bs*dsqrt(qs*(RI4+0.25d0*as*as))
      cl1=re1/re2
      cl2=cl1
      cl10=cl1
      cl20=cl2
c
      nn=1
      jj=1
      do 339 i=1,2
         rintv(i)=0.0d0
         rintv2(i)=0.0d0
 339  continue
      t=0.0
      count=0
      j=1
      rval(1)=r0
      out(1)=sngl(t)
      out(2)=sngl(y(1))
      out(3)=sngl(y(2))
      out(4)=sngl(y(3))
      eta1=y(1)
      w1=y(2)
      g1=y(3)
      vx1=y(5)
      vy1=y(6)
      xi1=y(7)
      e1=y(8)
      eta2=y(9)
      w2=y(10)
      g2=y(11)
      vx2=y(13)
      vy2=y(14)
      xi2=y(15)
      e2=y(16)
      rdis1=dsqrt(xi1*xi1+e1*e1)
      rdis2=dsqrt(xi2*xi2+e2*e2)
      rmom=RI2*eta1*eta1*w1*w1+cl1*g1*g1
      rmom=rmom+RI2*eta2*eta2*w2*w2+cl2*g2*g2
      write(4,61) out(1),eta1,w1,g1,vx1,xi1,e1,eta2,w2,g2,vx2,xi2,e2
c     &     rmom,rdis1,rdis2,alpha1,alpha2
c      write(6,62) out(1),eta1,w1,g1,vx1,xi1,e1,eta2,w2,g2,vx2,xi2,
c     &     e2,rmom,rdis1,rdis2


 30   continue ! Start of the loop


c      call halfval1(t,y)
c      call halfval2(t,y)
      call derivs(t,y,dydx)
      call RK4(y,dydx,nmax,t,h,y)
      count=count+1
```

```
      t=t+h
      j=j+1
      jj=jj+1
      out(1)=sngl(t)
      out(2)=sngl(y(1))
      out(3)=sngl(y(2))
      out(4)=sngl(y(3))
      if(count.eq.num) then
      eta1=y(1)
      w1=y(2)
      g1=y(3)
      vx1=y(5)
      vy1=y(6)
      xi1=y(7)
      e1=y(8)
      eta2=y(9)
      w2=y(10)
      g2=y(11)
      vx2=y(13)
      vy2=y(14)
      xi2=y(15)
      e2=y(16)
      rdis1=dsqrt(xi1*xi1+e1*e1)
      rdis2=dsqrt(xi2*xi2+e2*e2)
c

! write(6,*) 'Betafind t 2 ',t

      call betafind(eta1,w1,xi1,e1,eta2,w2,xi2,e2,bet1,bet2)
      call afind(eta1,w1,xi1,e1,eta2,w2,xi2,e2,bet1,bet2,alpha1,alpha2)

!      write(6,*) 'beta1=',bet1
!      write(6,*) 'beta2=',bet2
!      write(6,*) 'alpha1=',alpha1
!      write(6,*) 'alpha2=',alpha2
c
      count=0
      rmom=RI2*eta1*eta1*w1*w1+cl1*g1*g1
      rmom=rmom+RI2*eta2*eta2*w2*w2+cl2*g2*g2

!         write(6,*)
!         write(6,*) 'Main 1'

      write(55,61) out(1),bet1,alpha1,bet2,alpha2
      write(4,61) out(1),eta1,w1,g1,vx1,xi1,e1,eta2,w2,g2,vx2,xi2,e2
c     &           rmom,rdis1,rdis2,alpha1,alpha2
c      write(6,62) out(1),eta1,w1,g1,vx1,xi1,e1,eta2,w2,g2,vx2,xi2,
c     &   e2,rmom,rdis1,rdis2,alpha1,alpha2

!         write(6,*)
!         write(6,*) 'Main 2'

 60   format(15f17.6)
      end if
 61   format(f6.2,20f10.5)
 62   format(f6.2,20f10.5)
 620  format(F20.16,4X,I6)
 640  format(I6)

      if(t.lt.(tmax-0.5*h)) go to 30

         write(6,*)
!         write(6,*) 'Loop iter= ', iter, vx1, xi1
!         write(6,*) 'Loop iter= ', iter, vx2_s, xi1
!         write(6,*) y(1),y(2),y(3),y(4),y(5),y(6),y(7),y(8)
!         write(6,*) y(9),y(10),y(11),y(12),y(13),y(14),y(15),y(16)

         write(6,*) 'iter= ', iter,' V2 ', vx2_s,' X1 ', xi1, ' X2 ',xi2

!      write(6,*)
!      write(6,*) 'Loop iter= ',iter,xi1,vx2_s,tar_x,W_max_V,xi2
      if ((iter .le. 20).and.(dabs(tar_X-xi1)>1.0D-5)) then ! If the target value of X hasn't been
            vx2_s=PrimRad(Predict2(xi1,vx2_s,iter,tar_X,W_max_V,! achieved then loop around again.
     +            xi2,W_Min_V))
```

```
     if (vx2_s .ne. 0) then
iter=iter+1
goto 20
     else
         write(6,*) 'Beams too far apart to converge to target value'
     endif
      endif

c      write(6,*) 'cl1=',cl1
      te1=d1*d2*rmom00
      te2=d2*eta100*eta100*w100*w100+d1*eta200*eta200*w200*w200
      te3=d1*d2*rmomy00
      te4=d2*eta100*eta100*w100*w100+d1*eta200*eta200*w200*w200
      rip=(te1/(te2*RI2))**2.0d0+(te3/(te4*RI2))**2.0d0
      rip=dsqrt(rip)
!      write(6,*) 'output rip=',rip
!      write(6,*) 'eta100=',eta100
!      write(6,*) 'w100=',w100
!      write(6,*) 'eta200=',eta200
!      write(6,*) 'w200=',w200

        write(6,*)
        write(6,*) 'iter= ', iter,' V2 ', vx2_s,' X1 ', xi1, ' X2 ',xi2

c 776  continue
      close (4)
 54   stop
      end



      subroutine fixedfind(h0,eta00,w00,bet00)
c
c     This routine is used in determining l, the length of the
c     of the shelf radiation. It might not be needed for counter-
c     propagating nematicons. Only the initial values are needed to
c     calculate so it is only called once.
c
      implicit double precision (a-h,o-z)
      double precision bet1
c
      common/dblk/rnu,qs,d1,d2,AA,BB,cl1,cl2,cl10,cl20,W_adj,iflagr
c
      RI=dlog(2.0d0)/3.0d0+1.0d0/6.0d0
      RI1=2.0d0*0.915965594d0
      RI2=dlog(2.0d0)
      RI4=2.0d0*dlog(2.0d0)/3.0d0-1.0d0/6.0d0
      RI42=2.0d0*dlog(2.0d0)/15.0d0+1.0d0/60.0d0
      RI7=1.352314016d0
      RI8=16.0d0*dlog(2.0d0)/35.0d0-19.0d0/105.0d0
c
      as=dsqrt(2.0d0)*RI2/dsqrt(RI7)
      bs=dsqrt(2.0d0*RI2)
      j=0
c
c      write(6,*) 'Enter initial guess for w00'
c      read(5,*) w100
      w100=2.6d0
      OPEN(UNIT=44,FILE='energy.dat')
      CLOSE(44,STATUS='DELETE')
      OPEN(UNIT=44,FILE='energy.dat',STATUS='NEW')
      REWIND(44)
c
      w200=w100+0.01d0
c
      do 300 k=1,2000
      w1000=0.1d0+(20.0d0-0.1d0)*dble(k-1)/2000.0d0
      re1=qs*(RI4+0.25d0*as*as)*bs*bs*w1000*w1000
      re2=re1*re1
      re3=2.0d0*qs*rnu*(RI4+0.25d0*as*as)*(8.0d0*RI42+1.0d0)
      re3=re3*bs*bs*as*as*w1000*w1000
      re4=2.0d0*qs*(RI4+0.25d0*as*as)*as*as
      bet1=(re1+dsqrt(re2+re3))/re4
      bet1=dsqrt(bet1)
```

```
      re1=d1*RI*qs*(RI4+0.25d0*as*as)
      re1=re1*(as*as*bet1*bet1+bs*bs*w1000*w1000)**4.0d0
      re2=8.0d0*AA*AA*(as**4.0d0)*(bs**8.0d0)*(w1000**8.0d0)*bet1**2.0d0
      eta1=re1/re2
      eta1=dsqrt(eta1)
      re1=4.0d0*AA*as*as*bs*bs*eta1*eta1*w1000*w1000*bet1*bet1
      re2=8.0d0*rnu*RI42+4.0d0*qs*RI4*bet1*bet1+rnu+qs*as*as*bet1*bet1
      re2=re2*(as*as*bet1*bet1+bs*bs*w1000*w1000)
      alpha1=re1/re2
c
      den1=as*as*bet1*bet1+bs*bs*w1000*w1000
      te1=2.0d0*d1*RI*eta1*eta1
      te2=4.0d0*AA*as*as*bs*bs*alpha1*eta1*eta1*w1000*w1000
      te2=te2*bet1*bet1/den1
      hhh1=te1-te2-h0
c      write(44,301) w1000,hhh1
 300  continue
 301  format(2f28.12)
c
 100  continue
c
      re1=qs*(RI4+0.25d0*as*as)*bs*bs*w100*w100
      re2=re1*re1
      re3=2.0d0*qs*rnu*(RI4+0.25d0*as*as)*(8.0d0*RI42+1.0d0)
      re3=re3*bs*bs*as*as*w100*w100
      re4=2.0d0*qs*(RI4+0.25d0*as*as)*as*as
      bet1=(re1+dsqrt(re2+re3))/re4
      bet1=dsqrt(bet1)
      re1=d1*RI*qs*(RI4+0.25d0*as*as)
      re1=re1*(as*as*bet1*bet1+bs*bs*w100*w100)**4.0d0
      re2=8.0d0*AA*AA*(as**4.0d0)*(bs**8.0d0)*(w100**8.0d0)*bet1**2.0d0
      eta1=re1/re2
      eta1=dsqrt(eta1)
      re1=4.0d0*AA*as*as*bs*bs*eta1*eta1*w100*w100*bet1*bet1
      re2=8.0d0*rnu*RI42+4.0d0*qs*RI4*bet1*bet1+rnu+qs*as*as*bet1*bet1
      re2=re2*(as*as*bet1*bet1+bs*bs*w100*w100)
      alpha1=re1/re2
c
      den1=as*as*bet1*bet1+bs*bs*w100*w100
      te1=2.0d0*d1*RI*eta1*eta1
      te2=4.0d0*AA*as*as*bs*bs*alpha1*eta1*eta1*w100*w100
      te2=te2*bet1*bet1/den1
      hh1=te1-te2-h0
CNN      hh1=te1-te2+te4+te5-h0
c
      re1=qs*(RI4+0.25d0*as*as)*bs*bs*w200*w200
      re2=re1*re1
      re3=2.0d0*qs*rnu*(RI4+0.25d0*as*as)*(8.0d0*RI42+1.0d0)
      re3=re3*bs*bs*as*as*w200*w200
      re4=2.0d0*qs*(RI4+0.25d0*as*as)*as*as
      bet1=(re1+dsqrt(re2+re3))/re4
      bet1=dsqrt(bet1)
      re1=d1*RI*qs*(RI4+0.25d0*as*as)
      re1=re1*(as*as*bet1*bet1+bs*bs*w200*w200)**4.0d0
      re2=8.0d0*AA*AA*(as**4.0d0)*(bs**8.0d0)*(w200**8.0d0)*bet1**2.0d0
      eta1=re1/re2
      eta1=dsqrt(eta1)
      re1=4.0d0*AA*as*as*bs*bs*eta1*eta1*w200*w200*bet1*bet1
      re2=8.0d0*rnu*RI42+4.0d0*qs*RI4*bet1*bet1+rnu+qs*as*as*bet1*bet1
      re2=re2*(as*as*bet1*bet1+bs*bs*w200*w200)
      alpha1=re1/re2
c
      den1=as*as*bet1*bet1+bs*bs*w200*w200
      te1=2.0d0*d1*RI*eta1*eta1
      te2=4.0d0*AA*as*as*bs*bs*alpha1*eta1*eta1*w200*w200
      te2=te2*bet1*bet1/den1
      hh2=te1-te2-h0
CNN      hh2=te1-te2+te4+te5-h0
c
CNN      if(dabs(hh2-hh1).lt.1.0d-7) go to 200
      w1new=w200-(w200-w100)*hh2/(hh2-hh1)
c          write(6,*) 'w1=',w1new
c          write(6,*) 'beta1=',bet1
      diff=dabs(w1new-w200)
```

```
      j=j+1
      if(j.gt.100) stop
      if(diff.gt.1.0d-7) then
          w100=w200
          w200=w1new
          go to 100
      end if
c
 200  continue
      eta00=eta1
      w00=w1new
      bet00=bet1
c      write(6,*) 'eta00=',eta00
c      write(6,*) 'w00=',w00
c      write(6,*) 'alpha1=',alpha1
      return
      end



      subroutine afind(eta1,w1,xi1,e1,eta2,w2,xi2,e2,bet1,bet2,alpha1
c     Calculates alpha, the amplitude of the director.
c

     &  ,alpha2)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      parameter (nmax=16)
c
      common/dblk/rnu,qs,d1,d2,AA,BB,cl1,cl2,cl10,cl20,W_adj,iflagr
c

c         write(6,*)
c         write(6,*) 'Enter afind'

      RI=dlog(2.0d0)/3.0d0+1.0d0/6.0d0
      RI1=2.0d0*0.915965594d0
      RI2=dlog(2.0d0)
      RI4=2.0d0*dlog(2.0d0)/3.0d0-1.0d0/6.0d0
      RI42=2.0d0*dlog(2.0d0)/15.0d0+1.0d0/60.0d0
      RI7=1.352314016d0
      RI8=16.0d0*dlog(2.0d0)/35.0d0-19.0d0/105.0d0
c
      as=dsqrt(2.0d0)*RI2/dsqrt(RI7)
      bs=dsqrt(2.0d0*RI2)
c
      arg=(xi1-xi2)**2.0d0+(e1-e2)**2.0d0
      arg1=Width_calc(xi1,xi2,e1,e2,W_adj,w1,w2)
c      If (abs(arg-arg1) .gt. 1.0d-4) then
c      write(6,*) 'Afind 1 arg1=',arg1,' arg=',arg,' W=',W_adj,w1,w2
c      endif
      earg1=dexp(-arg/(as*as*bet1*bet1+bs*bs*w2*w2))
      earg2=dexp(-arg/(as*as*(bet1*bet1+bet2*bet2)))
      re1=AA*as*as*bs*bs*eta1*eta1*w1*w1*bet1*bet1
      re2=as*as*bet1*bet1+bs*bs*w1*w1
      te1=re1/re2
      re1=BB*as*as*bs*bs*eta2*eta2*w2*w2*bet1*bet1*earg1
      re2=as*as*bet1*bet1+bs*bs*w2*w2
      te2=re1/re2
      c11=te1+te2
      re1=1.0d0-arg/(as*as*(bet1*bet1+bet2*bet2))
      re1=re1*2.0d0*rnu*bet1*bet1*bet2*bet2*earg2
      re2=bet1*bet1+bet2*bet2
      te1=re1/(re2*re2)
      re1=qs*as*as*bet1*bet1*bet2*bet2*earg2
      re2=bet1*bet1+bet2*bet2
      te2=re1/re2
      c12=te1+te2
      earg1=dexp(-arg/(as*as*bet2*bet2+bs*bs*w1*w1))
      earg2=dexp(-arg/(as*as*(bet1*bet1+bet2*bet2)))
      re1=BB*as*as*bs*bs*eta2*eta2*w2*w2*bet2*bet2
      re2=as*as*bet2*bet2+bs*bs*w2*w2
      te1=re1/re2
      re1=AA*as*as*bs*bs*eta1*eta1*w1*w1*bet2*bet2*earg1
      re2=as*as*bet2*bet2+bs*bs*w1*w1
```

```
      te2=re1/re2
      c21=te1+te2
      c22=c12
c
      re1=2.0d0*c21*(2.0d0*rnu*RI42+qs*RI4*bet1*bet1)
      re1=re1-c12*c11
      re2=(2.0d0*rnu*RI42+qs*RI4*bet1*bet1)*(2.0d0*rnu*RI42+
     &     qs*RI4*bet2*bet2)
      re2=re2*4.0d0-c12*c12
      alpha2=re1/re2
      re1=c11-c12*alpha2
      re2=2.0d0*rnu*RI42+qs*RI4*bet1*bet1
      alpha1=re1/(2.0d0*re2)
c

c         write(6,*)
c         write(6,*) 'Exit afind'

      return
      end


      subroutine betafind(eta1,w1,xi1,e1,eta2,w2,xi2,e2,bet1,bet2)
c
c     Calculates beta, the width of the director.
c
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      parameter (nmax=16)
c
      common/dblk/rnu,qs,d1,d2,AA,BB,cl1,cl2,cl10,cl20,W_adj,iflagr
      common/guessblk/beti
c

c         write(6,*)
c         write(6,*) 'Enter Betafind'

      RI=dlog(2.0d0)/3.0d0+1.0d0/6.0d0
      RI1=2.0d0*0.915965594d0
      RI2=dlog(2.0d0)
      RI4=2.0d0*dlog(2.0d0)/3.0d0-1.0d0/6.0d0
      RI42=2.0d0*dlog(2.0d0)/15.0d0+1.0d0/60.0d0
      RI7=1.352314016d0
      RI8=16.0d0*dlog(2.0d0)/35.0d0-19.0d0/105.0d0
c
      as=dsqrt(2.0d0)*RI2/dsqrt(RI7)
      bs=dsqrt(2.0d0*RI2)
c
      bet10=beti
      bet11=beti+0.1d0
      bet20=beti
      bet21=beti+0.1d0
c
  100 continue

c      write(6,*)
c      write(6,*) 'Input params 2 = ',eta1,w1,xi1
c      write(6,*) e1,w2,xi2,e2
c      write(6,*) bet10,bet11,bet20,bet21
c      write(6,*) f100,f101,f110,f111
c      write(6,*) f200,f201,f210,f211

      call funfind1(eta1,w1,xi1,e1,eta2,w2,xi2,e2,bet10,bet20,f100)
c         write(6,*) 'Enter Betafind 1'
c      write (6,*) 'bet10 ',bet10,' bet20 ',bet20,' f100 ',f100
      call funfind1(eta1,w1,xi1,e1,eta2,w2,xi2,e2,bet11,bet20,f110)
c         write(6,*) 'Enter Betafind 2'
c      write (6,*) 'bet11 ',bet11,' bet20 ',bet20,' f110 ',f110
      call funfind1(eta1,w1,xi1,e1,eta2,w2,xi2,e2,bet10,bet21,f101)
c         write(6,*) 'Enter Betafind 3'
c      write (6,*) 'bet10 ',bet10,' bet21 ',bet21,' f101 ',f101
      call funfind2(eta1,w1,xi1,e1,eta2,w2,xi2,e2,bet10,bet20,f200)
c         write(6,*) 'Enter Betafind 4'
c      write (6,*) 'bet10 ',bet10,' bet20 ',bet20,' f200 ',f200
      call funfind2(eta1,w1,xi1,e1,eta2,w2,xi2,e2,bet11,bet20,f210)
```

```
c         write(6,*) 'Enter Betafind 5'
c      write (6,*) 'bet11 ',bet11,' bet20 ',bet20,' f210 ',f210
       call funfind2(eta1,w1,xi1,e1,eta2,w2,xi2,e2,bet10,bet21,f201)
c         write(6,*) 'Enter Betafind 6'
c      write (6,*) 'bet10 ',bet10,' bet21 ',bet21,' f201 ',f201
       call funfind1(eta1,w1,xi1,e1,eta2,w2,xi2,e2,bet11,bet21,f111)
c         write(6,*) 'Enter Betafind 7'
c      write (6,*) 'bet11 ',bet11,' bet21 ',bet21,' f111 ',f111
       call funfind2(eta1,w1,xi1,e1,eta2,w2,xi2,e2,bet11,bet21,f211)
c      write (6,*) 'bet11 ',bet11,' bet21 ',bet21,' f211 ',f211

c         write(6,*) 'Enter Betafind 8'

c      write(6,*)
c      write(6,*) 'Input params 3 = ',eta1,w1,xi1
c      write(6,*) e1,w2,xi2,e2

       f1x1=(f110-f100)/(bet11-bet10)
       f1x2=(f101-f100)/(bet21-bet20)
       f2x1=(f210-f200)/(bet11-bet10)
       f2x2=(f201-f200)/(bet21-bet20)

c      write(6,*)
c      write(6,*) 'Input params 4 = ',f1x1,f1x2,f2x1,f2x2

c
       det=f1x1*f2x2-f1x2*f2x1
c      write(6,*)
c      write(6,*) 'det=',det
       bet1new=bet11-(f2x2*f111-f1x2*f211)/det
       bet2new=bet21-(-f2x1*f111+f1x1*f211)/det
       if(dabs(bet1new).gt.1.d5) then
          write(6,*)
          write(6,*) 'Stop 1 bet1new=',bet1new
          stop
       end if
       if(dabs(bet2new).gt.1.d5) then
          write(6,*)
          write(6,*) 'Stop 2 bet2new=',bet2new
          stop
       end if
c
       diff=dsqrt((bet1new-bet11)**2.0d0+(bet2new-bet21)**2.0d0)
       if(diff.gt.1.0d-7) then
          bet10=bet11
          bet20=bet21
          bet11=bet1new
          bet21=bet2new
c         write(6,*)
c         write(6,*) 'diff=',diff,'bet1=',bet11,'bet2=',bet21
          go to 100
       end if
c
       bet1=bet1new
       bet2=bet2new
       beti_s=bet1new

c         write(6,*)
c         write(6,*) 'diff=',diff,'bet1=',bet11,'bet2=',bet21

c      write(6,*)
c      write(6,*) 'Exit betafind'

c
       return
       end

       subroutine funfind1(eta1,w1,xi1,e1,eta2,w2,xi2,e2,bet1,bet2,fff)
       implicit double precision (a-h,o-z)
c
       common/dblk/rnu,qs,d1,d2,AA,BB,cl1,cl2,cl10,cl20,W_adj,iflagr
c

c         write(6,*)
```

```
c         write(6,*) 'Enter Funfind1'

      RI=dlog(2.0d0)/3.0d0+1.0d0/6.0d0
      RI1=2.0d0*0.915965594d0
      RI2=dlog(2.0d0)
      RI4=2.0d0*dlog(2.0d0)/3.0d0-1.0d0/6.0d0
      RI42=2.0d0*dlog(2.0d0)/15.0d0+1.0d0/60.0d0
      RI7=1.352314016d0
      RI8=16.0d0*dlog(2.0d0)/35.0d0-19.0d0/105.0d0
c
      as=dsqrt(2.0d0)*RI2/dsqrt(RI7)
      bs=dsqrt(2.0d0*RI2)
c
      arg=(xi1-xi2)**2.0d0+(e1-e2)**2.0d0
      arg1=Width_calc(xi1,xi2,e1,e2,W_adj,w1,w2)
c      If (abs(arg-arg1) .gt. 1.0d-4) then
c      write(6,*) 'Funfind1 1 arg1=',arg1,' arg=',arg,W_adj,w1,w2
c      endif
      earg1=dexp(-arg/(as*as*bet1*bet1+bs*bs*w2*w2))
      earg2=dexp(-arg/(as*as*(bet1*bet1+bet2*bet2)))
      re1=AA*as*as*bs*bs*eta1*eta1*w1*w1*bet1*bet1
      re2=as*as*bet1*bet1+bs*bs*w1*w1
      te1=re1/re2
      re1=BB*as*as*bs*bs*eta2*eta2*w2*w2*bet1*bet1*earg1
      re2=as*as*bet1*bet1+bs*bs*w2*w2
      te2=re1/re2
      c11=te1+te2
      re1=1.0d0-arg/(as*as*(bet1*bet1+bet2*bet2))
      re1=re1*2.0d0*rnu*bet1*bet1*bet2*bet2*earg2
      re2=bet1*bet1+bet2*bet2
      te1=re1/(re2*re2)
      re1=qs*as*as*bet1*bet1*bet2*bet2*earg2
      re2=bet1*bet1+bet2*bet2
      te2=re1/re2
      c12=te1+te2
      earg1=dexp(-arg/(as*as*bet2*bet2+bs*bs*w1*w1))
      earg2=dexp(-arg/(as*as*(bet1*bet1+bet2*bet2)))
      re1=BB*as*as*bs*bs*eta2*eta2*w2*w2*bet2*bet2
      re2=as*as*bet2*bet2+bs*bs*w2*w2
      te1=re1/re2
      re1=AA*as*as*bs*bs*eta1*eta1*w1*w1*bet2*bet2*earg1
      re2=as*as*bet2*bet2+bs*bs*w1*w1
      te2=re1/re2
      c21=te1+te2
      c22=c12
c
      re1=2.0d0*c21*(2.0d0*rnu*RI42+qs*RI4*bet1*bet1)
      re1=re1-c12*c11
      re2=(2.0d0*rnu*RI42+qs*RI4*bet1*bet1)*(2.0d0*rnu*RI42+
     &     qs*RI4*bet2*bet2)
      re2=re2*4.0d0-c12*c12
      alpha2=re1/re2
      re1=c11-c12*alpha2
      re2=2.0d0*rnu*RI42+qs*RI4*bet1*bet1
      alpha1=re1/(2.0d0*re2)
c
      arg=(xi1-xi2)**2.0d0+(e1-e2)**2.0d0
      arg1=Width_calc(xi1,xi2,e1,e2,W_adj,w1,w2)
c      If (abs(arg-arg1) .gt. 1.0d-4) then
c      write(6,*) 'Funfind1 2 arg1=',arg1,' arg=',arg,W_adj,w1,w2
c      endif
      earg1=dexp(-arg/(as*as*bet1*bet1+bs*bs*w1*w1))
      earg2=dexp(-arg/(as*as*bet1*bet1+bs*bs*w2*w2))
      earg3=dexp(-arg/(as*as*(bet1*bet1+bet2*bet2)))
      te1=qs*RI4*alpha1
      re1=AA*as*as*(bs**4.0d0)*eta1*eta1*w1**4.0d0
      re2=as*as*bet1*bet1+bs*bs*w1*w1
      te2=re1/(re2*re2)
      re1=BB*as*as*(bs**4.0d0)*eta2*eta2*(w2**4.0d0)*earg2
      re2=as*as*bet1*bet1+bs*bs*w2*w2
      te3=re1/(re2*re2)
      re1=BB*(as**4.0d0)*bs*bs*eta2*eta2*w2*w2*bet1*bet1
      re1=re1*arg*earg2
      re2=as*as*bet1*bet1+bs*bs*w2*w2
```

```
      te4=re1/(re2*re2*re2)
      re1=(bet2*bet2-3.0d0*bet1*bet1)*arg/(as*as*
     &     (bet1*bet1+bet2*bet2))
      re1=-re1+bet2*bet2-bet1*bet1
      ttt1=bet1*bet1*arg*arg/((as**4.0d0)*(bet1*bet1+bet2*bet2)**2.0d0)
      re1=re1-ttt1
      re1=re1*2.0d0*rnu*alpha2*bet2*bet2*earg3
      re2=bet1*bet1+bet2*bet2
      te5=re1/(re2*re2*re2)
      re1=qs*as*as*alpha2*(bet2**4.0d0)*earg3
      re2=bet1*bet1+bet2*bet2
      te6=re1/(re2*re2)
      re1=qs*alpha2*bet1*bet1*bet2*bet2*arg*earg3
      re2=bet1*bet1+bet2*bet2
      te7=re1/(re2*re2*re2)
      fff=-te1+te2+te3+te4-te5-te6-te7
c

c         write(6,*)
c         write(6,*) 'Exit Funfind1'

      return
      end



      subroutine funfind2(eta1,w1,xi1,e1,eta2,w2,xi2,e2,bet1,bet2,fff)
      implicit double precision (a-h,o-z)
c
      common/dblk/rnu,qs,d1,d2,AA,BB,cl1,cl2,cl10,cl20,W_adj,iflagr
c

c         write(6,*)
c         write(6,*) 'Enter Funfind2'

      RI=dlog(2.0d0)/3.0d0+1.0d0/6.0d0
      RI1=2.0d0*0.915965594d0
      RI2=dlog(2.0d0)
      RI4=2.0d0*dlog(2.0d0)/3.0d0-1.0d0/6.0d0
      RI42=2.0d0*dlog(2.0d0)/15.0d0+1.0d0/60.0d0
      RI7=1.352314016d0
      RI8=16.0d0*dlog(2.0d0)/35.0d0-19.0d0/105.0d0
c
      as=dsqrt(2.0d0)*RI2/dsqrt(RI7)
      bs=dsqrt(2.0d0*RI2)
c
      arg=(xi1-xi2)**2.0d0+(e1-e2)**2.0d0
      arg1=Width_calc(xi1,xi2,e1,e2,W_adj,w1,w2)
c      If (abs(arg-arg1) .gt. 1.0d-4) then
c      write(6,*) 'Funfind2 1 arg1=',arg1,' arg=',arg,W_adj,w1,w2
c      endif
      earg1=dexp(-arg/(as*as*bet1*bet1+bs*bs*w2*w2))
      earg2=dexp(-arg/(as*as*(bet1*bet1+bet2*bet2)))
      re1=AA*as*as*bs*bs*eta1*eta1*w1*w1*bet1*bet1
      re2=as*as*bet1*bet1+bs*bs*w1*w1
      te1=re1/re2
      re1=BB*as*as*bs*bs*eta2*eta2*w2*w2*bet1*bet1*earg1
      re2=as*as*bet1*bet1+bs*bs*w2*w2
      te2=re1/re2
      c11=te1+te2
      re1=1.0d0-arg/(as*as*(bet1*bet1+bet2*bet2))
      re1=re1*2.0d0*rnu*bet1*bet1*bet2*bet2*earg2
      re2=bet1*bet1+bet2*bet2
      te1=re1/(re2*re2)
      re1=qs*as*as*bet1*bet1*bet2*bet2*earg2
      re2=bet1*bet1+bet2*bet2
      te2=re1/re2
      c12=te1+te2
      earg1=dexp(-arg/(as*as*bet2*bet2+bs*bs*w1*w1))
      earg2=dexp(-arg/(as*as*(bet1*bet1+bet2*bet2)))
      re1=BB*as*as*bs*bs*eta2*eta2*w2*w2*bet2*bet2
      re2=as*as*bet2*bet2+bs*bs*w2*w2
      te1=re1/re2
      re1=AA*as*as*bs*bs*eta1*eta1*w1*w1*bet2*bet2*earg1
```

```
      re2=as*as*bet2*bet2+bs*bs*w1*w1
      te2=re1/re2
      c21=te1+te2
      c22=c12
c
      re1=2.0d0*c21*(2.0d0*rnu*RI42+qs*RI4*bet1*bet1)
      re1=re1-c12*c11
      re2=(2.0d0*rnu*RI42+qs*RI4*bet1*bet1)*(2.0d0*rnu*RI42+
     &     qs*RI4*bet2*bet2)
      re2=re2*4.0d0-c12*c12
      alpha2=re1/re2
      re1=c11-c12*alpha2
      re2=2.0d0*rnu*RI42+qs*RI4*bet1*bet1
      alpha1=re1/(2.0d0*re2)
c
      arg=(xi1-xi2)**2.0d0+(e1-e2)**2.0d0
      arg1=Width_calc(xi1,xi2,e1,e2,W_adj,w1,w2)
c      If (abs(arg-arg1) .gt. 1.0d-4) then
c      write(6,*) 'Funfind2 2 arg1=',arg1,' arg=',arg,W_adj,w1,w2
c      endif
      earg1=dexp(-arg/(as*as*bet2*bet2+bs*bs*w2*w2))
      earg2=dexp(-arg/(as*as*bet2*bet2+bs*bs*w1*w1))
      earg3=dexp(-arg/(as*as*(bet2*bet2+bet1*bet1)))
      te1=qs*RI4*alpha2
      re1=BB*as*as*(bs**4.0d0)*eta2*eta2*w2**4.0d0
      re2=as*as*bet2*bet2+bs*bs*w2*w2
      te2=re1/(re2*re2)
      re1=AA*as*as*(bs**4.0d0)*eta1*eta1*(w1**4.0d0)*earg2
      re2=as*as*bet2*bet2+bs*bs*w1*w1
      te3=re1/(re2*re2)
      re1=AA*(as**4.0d0)*bs*bs*eta1*eta1*w1*w1*bet2*bet2
      re1=re1*arg*earg2
      re2=as*as*bet2*bet2+bs*bs*w1*w1
      te4=re1/(re2*re2*re2)
      re1=(bet1*bet1-3.0d0*bet2*bet2)*arg/(as*as*
     &     (bet2*bet2+bet1*bet1))
      re1=-re1+bet1*bet1-bet2*bet2
      ttt1=bet2*bet2*arg*arg/((as**4.0d0)*(bet2*bet2+bet1*bet1)**2.0d0)
      re1=re1-ttt1
      re1=re1*2.0d0*rnu*alpha1*bet1*bet1*earg3
      re2=bet2*bet2+bet1*bet1
      te5=re1/(re2*re2*re2)
      re1=qs*as*as*alpha1*(bet1**4.0d0)*earg3
      re2=bet2*bet2+bet1*bet1
      te6=re1/(re2*re2)
      re1=qs*alpha1*bet2*bet2*bet1*bet1*arg*earg3
      re2=bet2*bet2+bet1*bet1
      te7=re1/(re2*re2*re2)
      fff=-te1+te2+te3+te4-te5-te6-te7
c

c         write(6,*)
c         write(6,*) 'Exit Funfind2'

      return
      end


      subroutine halfval1(t,yy)
c
c     Not needed in this case, calculates radiation.
c
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      parameter (nmax=16)
      double precision rval(500000),rintv(2)
      double precision rval2(500000),rintv2(2)
      double precision yy(nmax)
      integer*4 jj,mike,iflag1
c
      common/param/eta100,w100,eta200,w200,r10,r20,h,mike
      common/flgblk/rval,rintv,rval2,rintv2,fa,iflag,jj,iflag1
      common/dblk/rnu,qs,d1,d2,AA,BB,cl1,cl2,cl10,cl20,W_adj,iflagr
      common/betblk/bet100,bet200
c
```

```
      eta1=yy(1)
      w1=yy(2)
      g1=yy(3)
      sig1=yy(4)
      vx1=yy(5)
      vy1=yy(6)
      xi1=yy(7)
      e1=yy(8)
      eta2=yy(9)
      w2=yy(10)
      g2=yy(11)
      sig2=yy(12)
      vx2=yy(13)
      vy2=yy(14)
      xi2=yy(15)
      e2=yy(16)
      rintv(1)=rintv(2)
c
      RI=dlog(2.0d0)/3.0d0+1.0d0/6.0d0
      RI1=2.0d0*0.915965594d0
      RI2=dlog(2.0d0)
      RI4=2.0d0*dlog(2.0d0)/3.0d0-1.0d0/6.0d0
      RI42=2.0d0*dlog(2.0d0)/15.0d0+1.0d0/60.0d0
      RI8=16.0d0*dlog(2.0d0)/35.0d0-19.0d0/105.0d0
c
CNN      cl1=0.5d0*(7.0d0*0.881373525*bet100)**2.0d0
      cl1new=0.5d0*(7.0d0*0.881373525*bet100)**2.0d0
c
      rr=eta1*eta1*w1*w1*RI2-eta100*eta100*w100*w100*RI2+cl1new*g1*g1
      rr=rr/cl1new
      rr=dsqrt(dabs(rr))
      cl12new=dsqrt(2.0d0*cl1new)
c
      pi=dacos(-1.d0)
      spi=dsqrt(pi)
      ee=dexp(1.0d0)
c
      rval(jj)=rr
c
      if(jj.gt.2) then
        rmike=0.0d0
        do 230 i=2,jj-1
            tau=(dfloat(i-1))*h
            arg=2.0d0*d1*(t-tau)/(cl12new*cl12new)
            rlo=dlog(arg)
            te1=0.25d0*0.25d0*rlo*rlo+3.0d0*pi*pi/16.0d0
            te2=pi*pi*rlo*rlo/16.0d0
            re1=pi*rlo/(32.0d0*ee*(te1*te1+te2)*(t-tau))
            rmike=rmike+h*rval(i)*dsqrt(2.0d0*pi)*re1
 230    continue
        arg=2.0d0*d1*t/(cl12new*cl12new)
        rlo=dlog(arg)
        te1=0.25d0*0.25d0*rlo*rlo+3.0d0*pi*pi/16.0d0
        te2=pi*pi*rlo*rlo/16.0d0
        re1=pi*rlo/(32.0d0*ee*(te1*te1+te2)*t)
        rmike=rmike+0.5d0*h*rval(1)*dsqrt(2.0d0*pi)*re1
c
        rintv(2)=rmike
c        rder=(rintv(jj)-rintv(jj-1))/h
      else
        rintv(2)=0.0d0
      end if
      if(jj.eq.2) then
        arg=2.0d0*d1*t/(cl12new*cl12new)
        rlo=dlog(arg)
        te1=0.25d0*0.25d0*rlo*rlo+3.0d0*pi*pi/16.0d0
        te2=pi*pi*rlo*rlo/16.0d0
        re1=pi*rlo/(32.0d0*ee*(te1*te1+te2)*t)
        rmike=h*rval(1)*dsqrt(2.0d0*pi)*re1
      end if
      return
      end
```

```
      subroutine halfval2(t,yy)
c
c     Not needed in this case, calculates radiation.
c
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      parameter (nmax=16)
      double precision rval2(500000),rintv2(2)
      double precision rval(500000),rintv(2)
      double precision yy(nmax)
      integer*4 jj,mike,iflag1
c
      common/param/eta100,w100,eta200,w200,r10,r20,h,mike
      common/flgblk/rval,rintv,rval2,rintv2,fa,iflag,jj,iflag1
      common/dblk/rnu,qs,d1,d2,AA,BB,cl1,cl2,cl10,cl20,W_adj,iflagr
      common/betblk/bet100,bet200
c
      eta1=yy(1)
      w1=yy(2)
      g1=yy(3)
      sig1=yy(4)
      vx1=yy(5)
      vy1=yy(6)
      xi1=yy(7)
      e1=yy(8)
      eta2=yy(9)
      w2=yy(10)
      g2=yy(11)
      sig2=yy(12)
      vx2=yy(13)
      vy2=yy(14)
      xi2=yy(15)
      e2=yy(16)
      rintv2(1)=rintv2(2)
c
      RI=dlog(2.0d0)/3.0d0+1.0d0/6.0d0
      RI1=2.0d0*0.915965594d0
      RI2=dlog(2.0d0)
      RI4=2.0d0*dlog(2.0d0)/3.0d0-1.0d0/6.0d0
      RI42=2.0d0*dlog(2.0d0)/15.0d0+1.0d0/60.0d0
      RI8=16.0d0*dlog(2.0d0)/35.0d0-19.0d0/105.0d0
c
CNN      cl2=0.5d0*(7.0d0*0.881373525*bet200)**2.0d0
      cl2new=0.5d0*(7.0d0*0.881373525*bet100)**2.0d0
c
      rr=eta2*eta2*w2*w2*RI2-eta200*eta200*w200*w200*RI2+cl2new*g2*g2
      rr=rr/cl2new
      rr=dsqrt(dabs(rr))
      cl22new=dsqrt(2.0d0*cl2new)
c
      pi=dacos(-1.d0)
      spi=dsqrt(pi)
      ee=dexp(1.0d0)
c
      rval2(jj)=rr
c
      if(jj.gt.2) then
          rmike=0.0d0
          do 230 i=2,jj-1
              tau=(dfloat(i-1))*h
              arg=2.0d0*d2*(t-tau)/(cl22new*cl22new)
              rlo=dlog(arg)
              te1=0.25d0*0.25d0*rlo*rlo+3.0d0*pi*pi/16.0d0
              te2=pi*pi*rlo*rlo/16.0d0
              re1=pi*rlo/(32.0d0*ee*(te1*te1+te2)*(t-tau))
              rmike=rmike+h*rval2(i)*dsqrt(2.0d0*pi)*re1
 230    continue
        arg=2.0d0*d2*t/(cl22new*cl22new)
        rlo=dlog(arg)
        te1=0.25d0*0.25d0*rlo*rlo+3.0d0*pi*pi/16.0d0
        te2=pi*pi*rlo*rlo/16.0d0
        re1=pi*rlo/(32.0d0*ee*(te1*te1+te2)*t)
        rmike=rmike+0.5d0*h*rval2(1)*dsqrt(2.0d0*pi)*re1
```

```
c
        rintv2(2)=rmike
c        rder=(rintv(jj)-rintv(jj-1))/h
      else
         rintv2(2)=0.0d0
      end if
      if(jj.eq.2) then
            arg=2.0d0*d2*t/(cl22new*cl22new)
            rlo=dlog(arg)
            te1=0.25d0*0.25d0*rlo*rlo+3.0d0*pi*pi/16.0d0
            te2=pi*pi*rlo*rlo/16.0d0
            re1=pi*rlo/(32.0d0*ee*(te1*te1+te2)*t)
            rmike=h*rval2(1)*dsqrt(2.0d0*pi)*re1
      end if
      return
      end




      subroutine derivs(t,y,dydx)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      parameter (nmax=16)
      double precision y(nmax),dydx(nmax),b(nmax,nmax),indx(nmax),d,t
      double precision bbb(nmax,nmax),in(nmax)
      double precision rval(500000),rintv(2)
      double precision rval2(500000),rintv2(2)
      integer*4 iflag,iflag1,jj
c
      common/param/eta100,w100,eta200,w200,r10,r20,h,mike
      common/flgblk/rval,rintv,rval2,rintv2,fa,iflag,jj,iflag1
      common/dblk/rnu,qs,d1,d2,AA,BB,cl1,cl2,cl10,cl20,W_adj,iflagr
c

c         write(6,*)
c         write(6,*) 'Enter Derivs'

      call matrix(y,t,b,dydx)
      do 78 i=1,nmax
         do 79 j=1,nmax
            bbb(i,j)=b(i,j)
  79     continue
  78  continue
      call LUDCMP(bbb,nmax,nmax,in,dd)
      do 67 j=1,nmax
         dd=dd*bbb(j,j)
  67  continue
c      write(6,*) 'det=',dd
      call LUDCMP(b,nmax,nmax,indx,d)
      call LUBKSB(b,nmax,nmax,indx,dydx)

c         write(6,*)
c         write(6,*) 'Exit Derivs'

      return
      end



      subroutine matrix(yy,t,r,rh)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      parameter (nmax=16)
      double precision r(nmax,nmax),rh(nmax),yy(nmax),t
      double precision rint1,rint2,rint3,g1,g2,rr
      double precision rval(500000),rintv(2)
      double precision rval2(500000),rintv2(2)
      integer*4 iflag,iflag1,jj, ic, jc
c
      common/param/eta100,w100,eta200,w200,r10,r20,h,mike
      common/flgblk/rval,rintv,rval2,rintv2,fa,iflag,jj,iflag1
      common/dblk/rnu,qs,d1,d2,AA,BB,cl1,cl2,cl10,cl20,W_adj,iflagr
      common/betblk/bet100,bet200
      common/momblk/xip
      common/guessblk/beti
```

```
c

c         write(6,*)
c         write(6,*) 'Enter Matrix'

      eta1=yy(1)
      w1=yy(2)
      g1=yy(3)
      sig1=yy(4)
      vx1=yy(5)
      vy1=yy(6)
      xi1=yy(7)
      e1=yy(8)
      eta2=yy(9)
      w2=yy(10)
      g2=yy(11)
      sig2=yy(12)
      vx2=yy(13)
      vy2=yy(14)
      xi2=yy(15)
      e2=yy(16)
c
      pi=dacos(-1.d0)
c
      RI=dlog(2.0d0)/3.0d0+1.0d0/6.0d0
      RI1=2.0d0*0.915965594d0
      RI2=dlog(2.0d0)
      RI4=2.0d0*dlog(2.0d0)/3.0d0-1.0d0/6.0d0
      RI42=2.0d0*dlog(2.0d0)/15.0d0+1.0d0/60.0d0
      RI7=1.352314016d0
      RI8=16.0d0*dlog(2.0d0)/35.0d0-19.0d0/105.0d0
c
      as=dsqrt(2.0d0)*RI2/dsqrt(RI7)
      bs=dsqrt(2.0d0*RI2)
c
CN      cl1=cl10
CN      cl2=cl20
CN      re1=2.0d0*as*RI1*RI1*dsqrt(rnu*RI42)*w1
CN      re2=3.0d0*bs*dsqrt(qs*(RI4+0.25d0*as*as))
CN      cl1=re1/re2
CN      cl2=cl1
c

c         write(6,*) 'Enter Matrix 1'

      call betafind(eta1,w1,xi1,e1,eta2,w2,xi2,e2,bet1,bet2)
      call afind(eta1,w1,xi1,e1,eta2,w2,xi2,e2,bet1,bet2,alpha1,alpha2)
      beti=bet1
c
      num1=2.0d0*qs*(RI4+(as*as*0.25d0)*bs*bs*w1
     &    *bet1*bet1)
      num2=num1 + rnu*((8.0d0*RI42)+1)*bs*bs*w1
      den1=qs*(RI4+(as*as*0.25d0))
      den2=den1*((4.0d0*as*as*bet1**3.0d0)
     &    -2.0d0*bs*bs*w1*w1*bet1)
      uu0=num2/den2
c
      num1=4.0d0*AA*as*as*bs**4.0d0*eta1*w1**4.0d0
      den1=qs*(RI4+(as*as*0.25d0))*((as*as*bet1*bet1
     &    + bs*bs*w1*w1)**2.0d0)
      uu1=num1/den1
c
      num1=8.0d0*AA*(as**4.0d0)*(bs**4.0d0)*eta1*eta1
     &    *(w1**3.0d0)*bet1
      den1=qs*(RI4+(as*as*0.25d0))*((as*as*bet1*bet1
     &    + bs*bs*w1*w1)**3.0d0)
      uu2=(bet1-w1*uu0)*(num1/den1)
c
      num1=d1*RI*(as*as*bet1*bet1+ bs*bs*w1*w1)
      num2=num1-AA*as*as*bs*bs*w1*w1*bet1*bet1
     &    *(2.0d0*alpha1+eta1*uu1)
      den1=AA*as*as*bs*bs*eta1*w1*bet1
      den2=den1/(as*as*bet1*bet1+ bs*bs*w1*w1)
      den3=2.0d0*as*as*alpha1*(bet1**3.0d0)
```

```
      den4=2.0d0*bs*bs*(w1**3.0d0)*alpha1*uu0
      den5=w1*bet1*uu2*(as*as*bet1*bet1
     &    + bs*bs*w1*w1)
      den6=den2*(den3+den4+den5)
      theta=num2/den6
c
      brack1=1-(2.0d0*eta1*theta/w1)
      pt1=alpha1*w1*bet1 + eta1*w1*bet1
     &    *(uu1+theta*uu2)
      num1=2.0d0*alpha1*eta1*(as*as*bet1*bet1
     &    - bs*bs*w1*w1)
      num2=num1*(bet1-uu0)*theta
      den1=as*as*bet1*bet1 + bs*bs*w1*w1
      pt2=num2/den1
      brack2=(pt1+pt2)/(alpha1*w1*bet1)
      bigtheta=d1*RI*(brack1-brack2)/2.0d0*w1*w1
c
c      num1=AA*as*as*bs*bs*alpha1*w1*w1*bet1*bet1
c     &    *(as*as*bet1*bet1+2.0d0*bs*bs*w1*w1)
c      den1=(as*as*bet1*bet1 + bs*bs*w1*w1)**2.0d0
c      brack1=num1/den1
c      num1=AA*as*as*bs*bs*alpha2*w1*w1*bet2*bet2
c     &    *(as*as*bet2*bet2+2.0d0*bs*bs*w1*w1)
c      den1=(as*as*bet2*bet2 + bs*bs*w1*w1)**2.0d0
c      brack2=num1/den1
c      sig1p=(brack1+brack2-2.0d0*RI*d1)/(RI2*w1*w1)
c
      num1=2.0d0*AA*as*as*bs*bs*alpha1*w1*w1*bet1*bet1
      num2=num1*(as*as*bet1*bet1-2.0d0*bs*bs*w1*w1)
      den1=RI2*((as*as*bet1*bet1+bs*bs*w1*w1)**2.0d0)
      sig1p=num2/den1
      cl1=-RI1*RI1*(w1*w1+2.0d0*eta1*w1*theta)
      cl1=cl1*sig1p/bigtheta
c      cl1=dsqrt(2.0d0*cl1)
      cl2=cl1

c         write(6,*) 'Enter Matrix 2'

Do ic=1,nmax
Do jc=1,nmax
r(ic,jc)=0.0d0
Enddo
Enddo
csi
      r(1,1)=2.0d0*RI2*eta1*w1*w1
      r(1,2)=2.0d0*RI2*eta1*eta1*w1
      r(1,3)=2.0d0*cl1*g1
c
      r(2,1)=RI1*w1*w1
      r(2,2)=2.0d0*RI1*eta1*w1
      r(2,4)=-cl1*g1
c
      r(3,1)=2.0d0*RI2*eta1*w1*w1*vx1
      r(3,2)=2.0d0*RI2*eta1*eta1*w1*vx1
      r(3,3)=2.0d0*cl1*g1*vx1
      r(3,5)=RI2*eta1*eta1*w1*w1+cl1*g1*g1
c
      r(4,1)=2.0d0*RI2*eta1*w1*w1*vy1
      r(4,2)=2.0d0*RI2*eta1*eta1*w1*vy1
      r(4,3)=2.0d0*cl1*g1*vy1
      r(4,6)=RI2*eta1*eta1*w1*w1+cl1*g1*g1
c
      r(5,7)=1.0d0
c
      r(6,8)=1.0d0
c
      r(7,3)=RI1
c
      r(8,4)=RI2*eta1*eta1*w1*w1
c
      r(9,9)=2.0d0*RI2*eta2*w2*w2
      r(9,10)=2.0d0*RI2*eta2*eta2*w2
      r(9,11)=2.0d0*cl2*g2
c
```

```
      r(10,9)=RI1*w2*w2
      r(10,10)=2.0d0*RI1*eta2*w2
      r(10,12)=-cl2*g2
c
      r(11,9)=2.0d0*RI2*eta2*w2*w2*vx2
      r(11,10)=2.0d0*RI2*eta2*eta2*w2*vx2
      r(11,11)=2.0d0*cl2*g2*vx2
      r(11,13)=RI2*eta2*eta2*w2*w2+cl2*g2*g2
c
      r(12,9)=2.0d0*RI2*eta2*w2*w2*vy2
      r(12,10)=2.0d0*RI2*eta2*eta2*w2*vy2
      r(12,11)=2.0d0*cl2*g2*vy2
      r(12,14)=RI2*eta2*eta2*w2*w2+cl2*g2*g2
c
      r(13,15)=1.0d0
c
      r(14,16)=1.0d0
c
      r(15,11)=RI1
c
      r(16,12)=RI2*eta2*eta2*w2*w2
c

      cl12=dsqrt(2.0d0*cl1)
      cll1=0.5d0*(7.0d0*0.881373525*bet100)**2.0d0
      rr1=eta1*eta1*w1*w1*RI2-eta100*eta100*w100*w100*RI2+cll1*g1*g1
      rr1=rr1/cll1
      rr1=dsqrt(dabs(rr1))
      if(rr1.ne.0.0d0) then
CNN      alp1=dabs(rintv(2))/(rr1*cl1)
      alp1=dabs(rintv(2))/(rr1*0.5d0*(7.0d0*0.881373525*bet100)**2.0d0)
      else
      alp1=0.0d0
      end if
      cl22=dsqrt(2.0d0*cl2)
      cll2=0.5d0*(7.0d0*0.881373525*bet200)**2.0d0
      rr2=eta2*eta2*w2*w2*RI2-eta200*eta200*w200*w200*RI2+cll2*g2*g2
      rr2=rr2/cll2
      rr2=dsqrt(dabs(rr2))
      if(rr2.ne.0.0d0) then
CNN      alp2=dabs(rintv2(2))/(rr2*cl2)
      alp2=dabs(rintv2(2))/(rr2*0.5d0*(7.0d0*0.881373525*bet200)**2.0d0)
      else
      alp2=0.0d0
      end if
c
CN      rh(1)=-dabs(2.0d0*d1*rr1*rr1*cl1*alp1)
      cl1new=0.5d0*(7.0d0*0.881373525*bet100)**2.0d0
      rh(1)=-dabs(2.0d0*d1*rr1*rr1*cl1new*alp1)
c
      rh(2)=-0.5d0*cl1*g1*d1*(vx1*vx1+vy1*vy1)
c
      x1mx2=xi1-xi2
      x1x2=(xi1-xi2)**2.0d0
      e1e2=(e1-e2)**2.0d0
      arg1=x1x2+e1e2
      arg=Width_calc(xi1,xi2,e1,e2,W_adj,w1,w2)
c      If (abs(arg-arg1) .gt. 1.0d-4) then
c      write(6,*) 'Matrix 1 arg1=',arg1,' arg=',arg,W_adj,w1,w2
c      endif
      earg1=dexp(-arg/(as*as*bet2*bet2+bs*bs*w1*w1))
      earg2=dexp(-arg/(as*as*bet1*bet1+bs*bs*w2*w2))
      earg3=dexp(-arg/(as*as*(bet1*bet1+bet2*bet2)))
c
      re1=2.0d0*AA*as*as*bs*bs*alpha2*eta1*eta1*w1*w1
      re1=re1*bet2*bet2*x1mx2*earg1
      te1=re1/(as*as*bet2*bet2+bs*bs*w1*w1)**2.0d0
      re1=2.0d0*BB*as*as*bs*bs*alpha1*eta2*eta2*w2*w2
      re1=re1*bet1*bet1*x1mx2*earg2
      te2=re1/(as*as*bet1*bet1+bs*bs*w2*w2)**2.0d0
      re1=4.0d0*rnu*alpha1*alpha2*bet1*bet1*bet2*bet2
      re1=re1*x1mx2*(2.0d0-arg/(bet1*bet1+bet2*bet2))
      te3=re1*earg3/(as*as*(bet1*bet1+bet2*bet2)**3.0d0)
      re1=2.0d0*qs*alpha1*alpha2*bet1*bet1*bet2*bet2
```

```
      re1=re1*x1mx2*earg3
      te4=re1/(bet1*bet1+bet2*bet2)**2.d00
      rh(3)=-te1-te2+te3+te4
c
      wt1=RI1*w1*w1*eta1**2-RI1*eta100*eta100*w100*w100
      wt2=cl1*g1**2
CN      fac=3.0d0*eta00/8.0d0
      fac=1.0d0/cl1
      rr=fac*(wt1+wt2)
      rr=dsqrt(dabs(rr))
      rloss=-8.0d0*cl1*alp1*rr*rr*(v1-xip)
CNN     rh(3)=rh(3)+rloss
c
      e1me2=e1-e2
      x1x2=(xi1-xi2)**2.0d0
      e1e2=(e1-e2)**2.0d0
      arg1=x1x2+e1e2
      arg=Width_calc(xi1,xi2,e1,e2,W_adj,w1,w2)
c      If (abs(arg-arg1) .gt. 1.0d-4) then
c      write(6,*) 'Matrix 2 arg1=',arg1,' arg=',arg,' W=',W_adj,w1,w2
c      endif
      earg1=dexp(-arg/(as*as*bet2*bet2+bs*bs*w1*w1))
      earg2=dexp(-arg/(as*as*bet1*bet1+bs*bs*w2*w2))
      earg3=dexp(-arg/(as*as*(bet1*bet1+bet2*bet2)))
c
      re1=2.0d0*AA*as*as*bs*bs*alpha2*eta1*eta1*w1*w1
      re1=re1*bet2*bet2*e1me2*earg1
      te1=re1/(as*as*bet2*bet2+bs*bs*w1*w1)**2.0d0
      re1=2.0d0*BB*as*as*bs*bs*alpha1*eta2*eta2*w2*w2
      re1=re1*bet1*bet1*e1me2*earg2
      te2=re1/(as*as*bet1*bet1+bs*bs*w2*w2)**2.0d0
      re1=4.0d0*rnu*alpha1*alpha2*bet1*bet1*bet2*bet2
      re1=re1*e1me2*(2.0d0-arg/(bet1*bet1+bet2*bet2))
      te3=re1*earg3/(as*as*(bet1*bet1+bet2*bet2)**3.0d0)
      re1=2.0d0*qs*alpha1*alpha2*bet1*bet1*bet2*bet2
      re1=re1*e1me2*earg3
      te4=re1/(bet1*bet1+bet2*bet2)**2.d00
      rh(4)=-te1-te2+te3+te4
c
      rh(5)=d1*vx1
c
      rh(6)=d1*vy1
c
      den1=as*as*bet1*bet1+bs*bs*w1*w1
      den2=as*as*bet2*bet2+bs*bs*w1*w1
c
      te1=d1*RI*eta1/(2.0d0*w1*w1)
      re1=AA*as*as*(bs**4.0d0)*alpha1*eta1*w1*w1
      te2=re1*bet1*bet1/den1**2.0d0
      re1=AA*as*as*(bs**4.0d0)*alpha2*eta1*w1*w1
      te3=re1*bet2*bet2*earg1/den2**2.0d0
      re1=AA*as*as*(bs**4.0d0)*alpha2*eta1*w1*w1
      te4=re1*bet2*bet2*arg*earg1/den2**3.0d0
      rh(7)=te1-te2-te3+te4
      rh(7)=rh(7)-2.0d0*RI1*alp1*d1*g1
c
      te1=0.5d0*RI2*eta1*eta1*w1*w1*(d1*vx1*vx1+d1*vy1*vy1)
      te2=d1*RI*eta1*eta1
      re1=AA*as*as*bs*bs*alpha1*eta1*eta1*w1*w1*bet1*bet1
      re1=re1*(as*as*bet1*bet1+2.0d0*bs*bs*w1*w1)
      te3=re1/den1**2.0d0
      re1=AA*as*as*bs*bs*alpha2*eta1*eta1*w1*w1*bet2*bet2
      re1=re1*(as*as*bet2*bet2+2.0d0*bs*bs*w1*w1)*earg1
      te4=re1/den2**2.0d0
      re1=AA*as*as*bs*bs*alpha2*eta1*eta1*(w1**4.0d0)
      re1=re1*bet2*bet2*arg*earg1
      te5=re1/den2**3.0d0
      rh(8)=te1-te2+te3+te4-te5
c
CN      rh(9)=-dabs(2.0d0*d2*rr2*rr2*cl2*alp2)
      cl2new=0.5d0*(7.0d0*0.881373525*bet200)**2.0d0
      rh(9)=-dabs(2.0d0*d2*rr2*rr2*cl2new*alp2)
c
      rh(10)=-0.5d0*cl2*g2*d2*(vx2*vx2+vy2*vy2)
```

```
c
      x2mx1=xi2-xi1
      x2x1=(xi2-xi1)**2.0d0
      e2e1=(e2-e1)**2.0d0
      arg1=x2x1+e2e1
      arg=Width_calc(xi1,xi2,e1,e2,W_adj,w1,w2)
c      If (abs(arg-arg1) .gt. 1.0d-4) then
c      write(6,*) 'Matrix 3 arg1=',arg1,' arg=',arg,W_adj,w1,w2
c      endif
      earg1=dexp(-arg/(as*as*bet1*bet1+bs*bs*w2*w2))
      earg2=dexp(-arg/(as*as*bet2*bet2+bs*bs*w1*w1))
      earg3=dexp(-arg/(as*as*(bet2*bet2+bet1*bet1)))
c
      re1=2.0d0*BB*as*as*bs*bs*alpha1*eta2*eta2*w2*w2
      re1=re1*bet1*bet1*x2mx1*earg1
      te1=re1/(as*as*bet1*bet1+bs*bs*w2*w2)**2.0d0
      re1=2.0d0*AA*as*as*bs*bs*alpha2*eta1*eta1*w1*w1
      re1=re1*bet2*bet2*x2mx1*earg2
      te2=re1/(as*as*bet2*bet2+bs*bs*w1*w1)**2.0d0
      re1=4.0d0*rnu*alpha2*alpha1*bet2*bet2*bet1*bet1
      re1=re1*x2mx1*(2.0d0-arg/(bet2*bet2+bet1*bet1))
      te3=re1*earg3/(as*as*(bet2*bet2+bet1*bet1)**3.0d0)
      re1=2.0d0*qs*alpha2*alpha1*bet2*bet2*bet1*bet1
      re1=re1*x2mx1*earg3
      te4=re1/(bet2*bet2+bet1*bet1)**2.d00
      rh(11)=-te1-te2+te3+te4
c
      wt1=RI1*w1*w1*eta1**2-RI1*eta100*eta100*w100*w100
      wt2=cl1*g1**2
c
CN      fac=3.0d0*eta00/8.0d0
      fac=1.0d0/cl2
      rr=fac*(wt1+wt2)
      rr=dsqrt(dabs(rr))
      rloss=-8.0d0*cl2*alp2*rr*rr*(v1-xip)
CNN      rh(11)=rh(11)+rloss
c
      e2me1=e2-e1
      x2x1=(xi2-xi1)**2.0d0
      e2e1=(e2-e1)**2.0d0
      arg1=x2x1+e2e1
      arg=Width_calc(xi1,xi2,e1,e2,W_adj,w1,w2)
c      If (abs(arg-arg1) .gt. 1.0d-4) then
c      write(6,*) 'Matrix 4 arg1=',arg1,' arg=',arg,W_adj,w1,w2
c      endif
      earg1=dexp(-arg/(as*as*bet1*bet1+bs*bs*w2*w2))
      earg2=dexp(-arg/(as*as*bet2*bet2+bs*bs*w1*w1))
      earg3=dexp(-arg/(as*as*(bet2*bet2+bet1*bet1)))
c
      re1=2.0d0*BB*as*as*bs*bs*alpha1*eta2*eta2*w2*w2
      re1=re1*bet1*bet1*e2me1*earg1
      te1=re1/(as*as*bet1*bet1+bs*bs*w2*w2)**2.0d0
      re1=2.0d0*AA*as*as*bs*bs*alpha2*eta1*eta1*w1*w1
      re1=re1*bet2*bet2*e2me1*earg2
      te2=re1/(as*as*bet2*bet2+bs*bs*w1*w1)**2.0d0
      re1=4.0d0*rnu*alpha2*alpha1*bet2*bet2*bet1*bet1
      re1=re1*e2me1*(2.0d0-arg/(bet2*bet2+bet1*bet1))
      te3=re1*earg3/(as*as*(bet2*bet2+bet1*bet1)**3.0d0)
      re1=2.0d0*qs*alpha2*alpha1*bet2*bet2*bet1*bet1
      re1=re1*e2me1*earg3
      te4=re1/(bet2*bet2+bet1*bet1)**2.d00
      rh(12)=-te1-te2+te3+te4
c
      rh(13)=d2*vx2
c
      rh(14)=d2*vy2
c
      den1=as*as*bet2*bet2+bs*bs*w2*w2
      den2=as*as*bet1*bet1+bs*bs*w2*w2
c
      te1=d2*RI*eta2/(2.0d0*w2*w2)
      re1=BB*as*as*(bs**4.0d0)*alpha2*eta2*w2*w2
      te2=re1*bet2*bet2/den1**2.0d0
      re1=BB*as*as*(bs**4.0d0)*alpha1*eta2*w2*w2
```

```
      te3=re1*bet1*bet1*earg1/den2**2.0d0
      re1=BB*as*as*(bs**4.0d0)*alpha1*eta2*w2*w2
      te4=re1*bet1*bet1*arg*earg1/den2**3.0d0
      rh(15)=te1-te2-te3+te4
      rh(15)=rh(15)-2.0d0*RI1*alp2*d2*g2
c
      te1=0.5d0*RI2*eta2*eta2*w2*w2*(d2*vx2*vx2+d2*vy2*vy2)
      te2=d2*RI*eta2*eta2
      re1=BB*as*as*bs*bs*alpha2*eta2*eta2*w2*w2*bet2*bet2
      re1=re1*(as*as*bet2*bet2+2.0d0*bs*bs*w2*w2)
      te3=re1/den1**2.0d0
      re1=BB*as*as*bs*bs*alpha1*eta2*eta2*w2*w2*bet1*bet1
      re1=re1*(as*as*bet1*bet1+2.0d0*bs*bs*w2*w2)*earg1
      te4=re1/den2**2.0d0
      re1=BB*as*as*bs*bs*alpha1*eta2*eta2*(w2**4.0d0)
      re1=re1*bet1*bet1*arg*earg1
      te5=re1/den2**3.0d0
      rh(16)=te1-te2+te3+te4-te5

c         write(6,*)
c         write(6,*) 'Exit Matrix'

c
      return
      end



      SUBROUTINE LUDCMP(A,N,NP,INDX,D)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      PARAMETER (nmax=16,TINY=1.0D-20)
      DOUBLE PRECISION A(NP,NP),INDX(NP),VV(NP)

c         write(6,*)
c         write(6,*) 'Enter LUDCMP'

      D=1.
      DO 12 I=1,N
        AAMAX=0.DO
        DO 11 J=1,N
          IF (DABS(A(I,J)).GT.AAMAX) AAMAX=DABS(A(I,J))
11      CONTINUE
        IF (AAMAX.EQ.0.DO) THEN
            WRITE(*,*) 'N = ', N, I, 'Singular matrix.'
            stop
c            READ(*,*)
        END IF
        VV(I)=1./AAMAX
12    CONTINUE
      DO 19 J=1,N
        IF (J.GT.1) THEN
          DO 14 I=1,J-1
            SUM=A(I,J)
            IF (I.GT.1)THEN
              DO 13 K=1,I-1
                SUM=SUM-A(I,K)*A(K,J)
13            CONTINUE
              A(I,J)=SUM
            ENDIF
14        CONTINUE
        ENDIF
        AAMAX=0.
        DO 16 I=J,N
          SUM=A(I,J)
          IF (J.GT.1)THEN
            DO 15 K=1,J-1
              SUM=SUM-A(I,K)*A(K,J)
15          CONTINUE
            A(I,J)=SUM
          ENDIF
          DUM=VV(I)*DABS(SUM)
          IF (DUM.GE.AAMAX) THEN
            IMAX=I
```

```
            AAMAX=DUM
          ENDIF
16      CONTINUE
        IF (J.NE.IMAX)THEN
          DO 17 K=1,N
            DUM=A(IMAX,K)
            A(IMAX,K)=A(J,K)
            A(J,K)=DUM
17        CONTINUE
          D=-D
          VV(IMAX)=VV(J)
        ENDIF
        INDX(J)=IMAX
        IF(J.NE.N)THEN
          IF(A(J,J).EQ.0.)A(J,J)=TINY
          DUM=1./A(J,J)
          DO 18 I=J+1,N
            A(I,J)=A(I,J)*DUM
18        CONTINUE
        ENDIF
19    CONTINUE
      IF(A(N,N).EQ.0.)A(N,N)=TINY

c         write(6,*)
c         write(6,*) 'Exit LUDCMP'

      RETURN
      END
C
      SUBROUTINE MPROVE(A,ALUD,N,NP,INDX,B,X)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      PARAMETER (nmax=16)
      DOUBLE PRECISION A(NMAX,NMAX),ALUD(NMAX,NMAX),INDX(NMAX)
      DOUBLE PRECISION B(NMAX),X(NMAX),R(NMAX)
      DOUBLE PRECISION SDP

      DO 12 I=1,N
        SDP=-B(I)
        DO 11 J=1,N
          SDP=SDP+A(I,J)*X(J)
11      CONTINUE
        R(I)=SDP
12    CONTINUE
      CALL LUBKSB(ALUD,N,NP,INDX,R)
      DO 13 I=1,N
        X(I)=X(I)-R(I)
13    CONTINUE
      RETURN
      END
C
      SUBROUTINE LUBKSB(A,N,NP,INDX,B)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DOUBLE PRECISION A(NP,NP),INDX(N),B(N)

c         write(6,*)
c         write(6,*) 'Enter LUBKSB'

      II=0
      DO 12 I=1,N
        LL=INDX(I)
        SUM=B(LL)
        B(LL)=B(I)
        IF (II.NE.0)THEN
          DO 11 J=II,I-1
            SUM=SUM-A(I,J)*B(J)
11        CONTINUE
        ELSE IF (SUM.NE.0.DO) THEN
          II=I
        ENDIF
        B(I)=SUM
12    CONTINUE
      DO 14 I=N,1,-1
        SUM=B(I)
        IF(I.LT.N)THEN
```

```
          DO 13 J=I+1,N
            SUM=SUM-A(I,J)*B(J)
13        CONTINUE
        ENDIF
        B(I)=SUM/A(I,I)
14    CONTINUE

c         write(6,*)
c         write(6,*) 'Exit LUBKSB'

      RETURN
      END
C
      SUBROUTINE RK4(Y,DYDX,N,X,H,YOUT)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      PARAMETER (nmax=16)
      DOUBLE PRECISION Y(N),DYDX(N),YOUT(N)
      DOUBLE PRECISION YT(NMAX),DYT(NMAX),DYM(NMAX)

c         write(6,*)
c         write(6,*) 'Enter RK4'

      HH=H*0.5D0
      H6=H/6.D0
      XH=X+HH
      DO 11 I=1,N
        YT(I)=Y(I)+HH*DYDX(I)
11    CONTINUE
      CALL DERIVS(XH,YT,DYT)
      DO 12 I=1,N
        YT(I)=Y(I)+HH*DYT(I)
12    CONTINUE
      CALL DERIVS(XH,YT,DYM)
      DO 13 I=1,N
        YT(I)=Y(I)+H*DYM(I)
        DYM(I)=DYT(I)+DYM(I)
13    CONTINUE
      CALL DERIVS(X+H,YT,DYT)
      DO 14 I=1,N
        YOUT(I)=Y(I)+H6*(DYDX(I)+DYT(I)+2.D0*DYM(I))
14    CONTINUE

c         write(6,*)
c         write(6,*) 'Exit RK4'

      RETURN
      END
*
************************************************************************
*
      Double Precision Function PrimRad(Radians)
!
!     Convert the input value to range -Pi/2, Pi/2
!     Input is an angle but this is periodic so
!     convert the angle to the range -Pi/2 to Pi/2.
!
      Double precision  Radians, Pi

      Pi=3.1415926535897932385
      PrimRad=Radians-Pi/2.0D00 *int(Radians/Pi*2.0D00)
      Return
      End

*
************************************************************************
*
      Double Precision Function Width_calc(X1,X2,Y1,Y2,fact,w1,w2)
!
!     Adjust the calculation of the width between the two beams
!     by subtracting a factor times the widths of the beams
!
      Double precision X1,X2,Y1,Y2,fact,w1,w2
      Double precision Wrk1, Wrk2, Wrk3, Wrk4, W_x, W_y
```

```
            Wrk1 = dabs(X1-X2)
            Wrk2 = fact*(w1+w2)
            If (Wrk1 .gt. Wrk2) then
               W_x = Wrk1-Wrk2
            else
               W_x = 0.0D0
            Endif

            Wrk3 = dabs(Y1-Y2)
            Wrk4 = fact*(w1+w2)
            If (Wrk3 .gt. Wrk4) then
               W_y = Wrk3-Wrk4
            else
               W_y = 0.0D0
            Endif

            Width_calc = W_x*W_x+W_y*W_y

      Return
      End
```

The programs to calculate the results for the extended particle model modulation equations is:

```
!     NLS_anal solves the differential equations that result when the path of two laser beams
!     is predicted using classical analytical mechanics. The distance between the two is
!     calculated using the initial conditions and a differential equation derived using
!     classical mechanics. The path of the upper laser beam is then calculated using the initial
!     conditions and the boundary condition that the laser beam is to end at a prescribed position.
!     The NLSHEAT program has been used to read in all the parameters for the laser beams. These parameters
!     are used to setup the potential function.

      PROGRAM NLS_anal_0
      implicit none
      Type Input_data
         Integer :: N_x, N_y, N_steps, NS_x, NS_y, Imid, IDebug, Ncont, Reg_w1
         Real (kind=10) :: T_step, T_initial, T_final
         Real (kind=10) :: Space_x, Space_y
         Real (kind=10) :: X_min, X_max, Y_min, Y_max
         Real (kind=10) :: Amp_1, Amp_2, Width_1, Width_2
         Real (kind=10) :: X_d1, Y_d1, Vel_1, Target_x
         Real (kind=10) :: X_d2, Y_d2, Vel_2
         Real (kind=10) :: Diff_1, Diff_2, AAAA, BBBB
         Real (kind=10) :: Nu, Theta, qs, R_phase, EPS, Om
         Real (kind=10) :: N_dp1, N_dp2, Damp_1, Damp_2, SRF
         Real (kind=10) :: Phi, Pi
         Real (kind=10) :: alpha1, alpha2, beta1, beta2, grad, inter, w1, w2, Reg_amt
      End Type
      Type (input_data) Input_rec

      Integer :: iter, Final_iter, i_lim, M_cross, M_bnd, it_2
      Integer, Parameter :: Max_iter=20000
      Real (kind=10) :: work_1, Potential, Xi_1, Xi_2, ODE_func, Predict, PrimRad
!
!     The following array variables are for Xi_u, Xi_v and the distance between them, Xi_u - Xi_v
!
      Real (kind=10), Dimension(Max_iter) :: W_xi_1, W_xi_2, W_x, W_z
      Real (kind=10) :: W_mass_1, W_mass_2, W_I2, W_energy
      Real (kind=10) :: W_K1, W_K2, W_K3, W_K4, W_h, WS_xi, xpos1, xpos2
      Real (kind=10) :: W_pot, W_KE, W_deriv, W_mult_const, W_add_const, v2, Target_X1, W_factor

      call Wtime('Start')

      CALL SETUP(Input_rec)
      call BetaFind(Input_rec)
      Call Write_input(Input_rec)

      v2=Input_rec%Vel_2

      Target_X1=Input_rec%Target_x
      W_I2 = log(2.0_10)
```

```
      Final_iter = Input_rec%N_steps

      do i_lim = 1, Final_iter

      OPEN (90,FILE='nlsanal_v2.dat',STATUS='UNKNOWN')
      CLOSE (90,STATUS='DELETE')
      OPEN (90,FILE='nlsanal_v2.dat',STATUS='UNKNOWN')
      M_cross = 0
      M_bnd = 0
      iter=1

              call BetaFind(Input_rec)
!              Call Write_input(Input_rec)

      W_xi_1(iter) = Input_rec%X_d1
      W_xi_2(iter) = Input_rec%X_d2
      W_x(iter)    = Input_rec%X_d1 - Input_rec%X_d2
              W_z(iter)    = Input_rec%T_initial
      W_mass_1 = W_I2 * Input_rec%Amp_1**2 * Input_rec%Width_1**2
      W_mass_2 = W_I2 * Input_rec%Amp_2**2 * Input_rec%Width_2**2
              W_mass   = W_mass_1
      W_KE = 0.5_10 * W_mass * (tan(Input_rec%Vel_1) - tan(v2))**2
      W_pot = 2.0_10 * Potential(W_x(iter), Input_rec, W_xi_1(iter), W_xi_2(iter))
              W_factor = abs(W_KE) / abs (W_pot)
      W_energy = W_KE + W_pot * W_factor

      W_mult_const = tan(Input_rec%Vel_1) + tan(v2)
      W_add_const  = Input_rec%X_d1 + Input_rec%X_d2

      print*,'mass ',W_mass,' KE = ',W_KE, ' Potential E = ',W_pot,' Energy = ',W_energy, ' at z=0'
      print*,'W_mult_const ',W_mult_const,' W_add_const ',W_add_const, ' v2', v2, ' W factor', W_factor

      W_h = Input_rec%T_step
              WRITE (90,910) W_z(iter), W_x(iter), W_xi_1(iter), W_xi_2(iter), W_pot
      do iter = 1, Input_rec%N_steps-1
 WS_xi = W_x(iter)
 W_K1 = ODE_func(W_mass_1, W_energy, WS_xi, Input_rec,iter, v2, W_xi_1(iter), W_xi_2(iter))
 W_K2 = ODE_func(W_mass_1, W_energy, WS_xi+W_K1/2.0_10, Input_rec,iter+1, v2, W_xi_1(iter), W_xi_2(iter))
 W_K3 = ODE_func(W_mass_1, W_energy, WS_xi+W_K2/2.0_10, Input_rec,iter+1, v2, W_xi_1(iter), W_xi_2(iter))
 W_K4 = ODE_func(W_mass_1, W_energy, WS_xi+W_K3, Input_rec,iter+1, v2, W_xi_1(iter), W_xi_2(iter))
 W_x(iter+1) = W_x(iter) + (W_K1 + 2.0_10 * W_K2 + 2.0_10 * W_K3 + W_K4) * W_h / 6.0_10
 W_z(iter+1) = (iter+1) * W_h
 xpos1 = 0.5_10 * (W_x(iter+1) + W_mult_const * W_z(iter+1) + W_add_const)
 xpos2 = 0.5_10 * (W_mult_const * W_z(iter+1) + W_add_const - W_x(iter+1))
 W_xi_1(iter+1) = xpos1
 W_xi_2(iter+1) = xpos2
 if (xpos2 .gt. xpos1) then
    M_cross=1
 end if
!  W_pot = Potential(WS_xi, Input_rec, W_xi_1(iter), W_xi_2(iter))
 W_pot = Potential(W_x(iter+1), Input_rec, xpos1, xpos2) * W_factor
                it_2 = iter+1
 if (mod(it_2,100).eq.0) then
    print 920,' Z ',W_z(it_2),' X ',W_x(it_2),' Xi_1 ',W_xi_1(it_2),' Xi_2 ',W_xi_2(it_2),' Pot E ',W_pot,' ODE ',W_K4
 end if
 WRITE (90,910) W_z(it_2), W_x(it_2), W_xi_1(it_2), W_xi_2(it_2), W_pot
             end do
!
!     If the absolute error between the X position calculated and the target position is greater than 1D-5
!     then call the Predict function to calculate the new angle. The PrimRad function ensures that the angle is
!     between Pi/2 and -Pi/2. The program has 20 chances to do this. If the laser beam does not form a soliton
!     or the two beams are very close then it may not be possible to achieve the target value of X.
!
      WRITE (6,1300) i_lim,xpos1,v2,abs(Target_X1-xpos1),xpos2,M_cross, M_bnd
      if (i_lim .le. 20) then
         If (abs(Target_X1-xpos1)>1.0D-5) then
            v2=Predict(xpos1,v2,i_lim,Target_X1,Xpos2,M_cross,M_Bnd)
                 else
                    exit
         endif
      else
         write(6,*) 'Beam can not converge to target value'
         exit
      endif
```

```
      Close (90)

      End do

      call Wtime('End')

 200  FORMAT (I10,14X,I6)
 220  FORMAT (F20.16,4X,I6)
 240  FORMAT (8X,I10,12X,A40)
 260  FORMAT (8X,F24.16,2X,A40)
 910  FORMAT (1X,F10.2,2X,4F28.12)
 920  FORMAT ( 1X,A8,F6.2,3(A8,F9.4),2(A8,ES16.8) )
1300  Format (I4,' Xpos1 ',F12.8,' V2 ',F12.8,' Abs error ',F12.8,' Xpos2 ',F12.8,' M_cross ',I4,' M_bnd ',I4)

      END Program NLS_anal_0
!*
!**********************************************************************
!*
      SUBROUTINE SETUP(Input_rec)
      implicit none
      Type Input_data
         Integer :: N_x, N_y, N_steps, NS_x, NS_y, Imid, IDebug, Ncont, Reg_w1
         Real (kind=10) :: T_step, T_initial, T_final
         Real (kind=10) :: Space_x, Space_y
         Real (kind=10) :: X_min, X_max, Y_min, Y_max
         Real (kind=10) :: Amp_1, Amp_2, Width_1, Width_2
         Real (kind=10) :: X_d1, Y_d1, Vel_1, Target_x
         Real (kind=10) :: X_d2, Y_d2, Vel_2
         Real (kind=10) :: Diff_1, Diff_2, AAAA, BBBB
         Real (kind=10) :: Nu, Theta, qs, R_phase, EPS, Om
         Real (kind=10) :: N_dp1, N_dp2, Damp_1, Damp_2, SRF
         Real (kind=10) :: Phi, Pi
         Real (kind=10) :: alpha1, alpha2, beta1, beta2, grad, inter, w1, w2, Reg_amt
      End Type
      Type(Input_data), intent(out) :: Input_rec

      OPEN (15,FILE='nls_anal.in',STATUS='OLD')
      REWIND (15)
!     N = number of x spatial points
      READ (15,200) Input_rec % N_x
!     NY = number of y spatial points
      READ (15,200) Input_rec % N_y
!     dt = timestep
      READ (15,220) Input_rec % T_step
!     t0 = start time
      READ (15,220) Input_rec % T_initial
!     tf = final time
      READ (15,220) Input_rec % T_final
!     nsteps = number of time steps to take
      Input_rec%N_steps = Int((Input_rec%T_final - Input_rec%T_initial) / Input_rec%T_step+0.5_10)
!     space = x interval
      READ (15,220) Input_rec % Space_x
!     spacey = y interval
      READ (15,220) Input_rec % Space_y
!     NSX =  x spacing for plotting 3D graph
      READ (15,200) Input_rec % NS_x
!     NSY =  y spacing for plotting 3D graph
      READ (15,200) Input_rec % NS_y
!     xmin = minimum x point for plot
      READ (15,220) Input_rec % X_min
!     xmax = maximum x point for plot
      READ (15,220) Input_rec % X_max
!     ymin = minimum y point for plot
      READ (15,220) Input_rec % Y_min
!     ymax = maximum y point for plot
      READ (15,220) Input_rec % Y_max
!     a1 = initial amplitude for u
      READ (15,220) Input_rec % Amp_1
!     a2 = initial amplitude for v
      READ (15,220) Input_rec % Amp_2
!     w1 = inital width for u
      READ (15,220) Input_rec % Width_1
!     w2 = inital width for v
      READ (15,220) Input_rec % Width_2
```

```
!     phi = initial phase difference
      READ (15,220) Input_rec % phi
!     xd = x position for u
      READ (15,220) Input_rec % X_d1
!     yd = y offset position for u
      READ (15,220) Input_rec % Y_d1
!     V = velocity for u
      READ (15,220) Input_rec % Vel_1
!     xd2 = x position for v
      READ (15,220) Input_rec % X_d2
!     yd2 = y offset position for v
      READ (15,220) Input_rec % Y_d2
!     V2 = velocity for v
      READ (15,220) Input_rec % Vel_2
!     diff1 = diffraction coefficient for u
      READ (15,220) Input_rec % Diff_1
!     diff2 = diffraction coefficient for v
      READ (15,220) Input_rec % Diff_2
!     AAAA = A for u
      READ (15,220) Input_rec % AAAA
!     BBBB = B for u
      READ (15,220) Input_rec % BBBB
!     rnu = nu
      READ (15,220) Input_rec % Nu
!     th0 = theta_{0}
      READ (15,220) Input_rec % Theta
      Input_rec % Pi = 3.1415926535897932385_10
!     qs = q_{s}
      READ (15,220) Input_rec % qs
!     r = rotating phase factor
      READ (15,220) Input_rec % R_phase
!     eps = amplitude of periodic forcing
      READ (15,220) Input_rec % EPS
!     om = frequency of periodic forcing, omega
      READ (15,220) Input_rec % OM
!     x distance from end for start of damping's polynomial cutoff
      READ (15,220) Input_rec % N_dp1
!     x distance from end for end of damping's polynomial cutoff
      READ (15,220) Input_rec % N_dp2
!     height of damping profile
      READ (15,220) Input_rec % Damp_1
!     coefficient of x inside of damping profile (1/width)
      READ (15,220) Input_rec % Damp_2
!     plot frequency for midpoint data
      READ (15,200) Input_rec % Imid
!     number of surfer plots to write out
      READ (15,220) Input_rec % SRF
!     flag to decide to plot damping or not
      READ (15,200) Input_rec % IDEBUG
!     ncont = 0 for initial start, 1 for continuing from last output file      Type Input_data
      READ (15,200) Input_rec % NCONT
!     Target X position. Want to get a solution that ends at this X position
      READ (15,220) Input_rec % Target_x
!     Width_const used to allow for width of beams on potential between the beams
!     READ (15,220) Input_rec % Width_const
      CLOSE (15)

!     Initial values for the apmlitude and width of the director

      Input_rec % alpha1 = 1.0_10
      Input_rec % alpha2 = 1.0_10
      Input_rec % beta1  = 10.0_10
      Input_rec % beta2  = 10.0_10

!      Call Write_input(Input_rec)

 200  FORMAT (I10,14X,I6)
 220  FORMAT (F20.16,4X,I6)
 240  FORMAT (8X,I10,16X,A40)
 260  FORMAT (8X,F24.16,2X,A40)

      End subroutine
!*
!*********************************************************************
```

```
!*
      Subroutine Write_input(Input_rec)
      implicit none

      Type Input_data
         Integer :: N_x, N_y, N_steps, NS_x, NS_y, Imid, IDebug, Ncont, Reg_w1
         Real (kind=10) :: T_step, T_initial, T_final
         Real (kind=10) :: Space_x, Space_y
         Real (kind=10) :: X_min, X_max, Y_min, Y_max
         Real (kind=10) :: Amp_1, Amp_2, Width_1, Width_2
         Real (kind=10) :: X_d1, Y_d1, Vel_1, Target_x
         Real (kind=10) :: X_d2, Y_d2, Vel_2
         Real (kind=10) :: Diff_1, Diff_2, AAAA, BBBB
         Real (kind=10) :: Nu, Theta, qs, R_phase, EPS, Om
         Real (kind=10) :: N_dp1, N_dp2, Damp_1, Damp_2, SRF
         Real (kind=10) :: Phi, Pi
         Real (kind=10) :: alpha1, alpha2, beta1, beta2, grad, inter, w1, w2, Reg_amt
      End Type
      Type(Input_data), intent(in) :: Input_rec

!      IF (IDEBUG .EQ. 1) THEN
       WRITE (6,*)
       WRITE (6,240) Input_rec % N_x,'... number of x spatial points'
       WRITE (6,240) Input_rec % N_y,'... number of y spatial points'
       WRITE (6,260) Input_rec % T_step,'... time step'
       WRITE (6,260) Input_rec % T_initial,'... start time'
       WRITE (6,260) Input_rec % T_final,'... stop time'
       WRITE (6,240) Input_rec % N_steps,'... number Runge-Kutta step'
       WRITE (6,260) Input_rec % Space_x,  '... x spatial interval'
       WRITE (6,260) Input_rec % Space_y, '... y spatial interval'
       WRITE (6,260) Input_rec % X_min,   '... minimum x point for plot'
       WRITE (6,260) Input_rec % X_max,   '... maximum x point for plot'
       WRITE (6,260) Input_rec % Y_min,   '... minimum y point for plot'
       WRITE (6,260) Input_rec % Y_max,   '... maximum y point for plot'
       IF (input_rec%NCONT .EQ. 0) THEN
           WRITE (6,260) Input_rec % Amp_1,'... initial amplitude u'
           WRITE (6,260) Input_rec % Width_1,'... initial width for u'
           WRITE (6,260) Input_rec % Amp_2,'... initial amplitude for v'
           WRITE (6,260) Input_rec % Width_2,'... initial width for v'
           WRITE (6,260) Input_rec % X_d1,'... x centre position for u'
           WRITE (6,260) Input_rec % Y_d1,'... y centre position for u'
           WRITE (6,260) Input_rec % Vel_1,'... velocity for u'
           WRITE (6,260) Input_rec % X_d2,'... x centre position for v'
           WRITE (6,260) Input_rec % Y_d2,'... y centre position for v'
           WRITE (6,260) Input_rec % Vel_2,'... velocity for v'
           WRITE (6,260) Input_rec % diff_1,'... diffraction coeff u'
           WRITE (6,260) Input_rec % AAAA,'A for u'
           WRITE (6,260) Input_rec % diff_2,'... diffraction coeff v'
           WRITE (6,260) Input_rec % BBBB,'B for v'
           WRITE (6,260) Input_rec % Nu,'... nu'
           WRITE (6,260) Input_rec % qs,'... q_{s}'
       ENDIF
       WRITE (6,260) Input_rec % EPS,'... periodic forcing amplitude'
       WRITE (6,260) Input_rec % OM,'... periodic forcing frequency'
       WRITE (6,260) Input_rec % N_dp1,'... start poly damping cutoff'
       WRITE (6,260) Input_rec % N_dp2,'... end poly damping cutoff'
       WRITE (6,260) Input_rec % Damp_1,'... damping layer amplitude'
       WRITE (6,260) Input_rec % Damp_2,'... damping layer decay rate'
       WRITE (6,260) Input_rec % SRF,'... number of surfer profiles'
       WRITE (6,240) Input_rec % IDEBUG,'... debug (1=yes, 0=no)'
       WRITE (6,240) Input_rec % NCONT,'... continuation (1=yes, 0=no)'
       Write (6,260) Input_rec % Target_X,'...target value of X'
       Write (6,260) Input_rec % alpha1,'...amplitude of beam 1'
       Write (6,260) Input_rec % alpha2,'...amplitude of beam 2'
       Write (6,260) Input_rec % beta1,'...width of beam 1'
       Write (6,260) Input_rec % beta2,'...width of beam 2'
       Write (6,280)

 200  FORMAT (I10,14X,I6)
 220  FORMAT (F20.16,4X,I6)
 240  FORMAT (8X,I10,16X,A40)
 260  FORMAT (8X,F24.16,2X,A40)
 280  FORMAT (8X)
```

```
      END subroutine Write_input
!*
!**********************************************************************
!*
      Subroutine Wtime(A)
!
!     This subroutine prints out the time with the message A. So
!     calling Wtime('Start') will print out 'Start' and the time.
!
      INTEGER*4 Now(3)
      Character (*) A

      call itime(now)
      write(*,1000) A, ' ,time=', now(1), now(2), now(3)

1000  format(A20,A7, i2.2, ':', i2.2, ':', i2.2)
      End subroutine
!*
!**********************************************************************
!*
      Real (kind=10) Function PrimRad(Radians)
!
!     Convert the input value to range -Pi/2, Pi/2
!     Input is an angle but this is periodic so
!     convert the angle to the range -Pi/2 to Pi/2.
!
      Real (kind=10)  Radians, Pi, wk1

      Pi=3.1415926535897932385_10
      wk1=Radians-Pi/2.0_10 *int(Radians/Pi*2.0_10)
      if (abs(wk1) .gt. 1.5_10) then
         wk1 = sign(1.5_10,wk1)
      endif
      print*,'Rad',Radians,'wk1',wk1
      PrimRad=wk1
      Return
      End
!*
!**********************************************************************
!*
      Real (kind=10) Function Potential(X, Input_rec, Xpos1, Xpos2)
      implicit none
!
!     Given Xi_1, Xi_2 calculate the attraction between 2 laser beams
!     in a liquid crystal. Xi_1 and Xi_2 are the postions in the x plane
!     of the 2 laser beams. All the information relating to the laser beams
!     in the liquid crystal is contained in Input_rec (the parameters entered
!     at the start of the program).
!

      Type Input_data
         Integer :: N_x, N_y, N_steps, NS_x, NS_y, Imid, IDebug, Ncont, Reg_w1
         Real (kind=10) :: T_step, T_initial, T_final
         Real (kind=10) :: Space_x, Space_y
         Real (kind=10) :: X_min, X_max, Y_min, Y_max
         Real (kind=10) :: Amp_1, Amp_2, Width_1, Width_2
         Real (kind=10) :: X_d1, Y_d1, Vel_1, Target_x
         Real (kind=10) :: X_d2, Y_d2, Vel_2
         Real (kind=10) :: Diff_1, Diff_2, AAAA, BBBB
         Real (kind=10) :: Nu, Theta, qs, R_phase, EPS, Om
         Real (kind=10) :: N_dp1, N_dp2, Damp_1, Damp_2, SRF
         Real (kind=10) :: Phi, Pi
         Real (kind=10) :: alpha1, alpha2, beta1, beta2, grad, inter, w1, w2, Reg_amt
      End Type

      Real (kind=10), Intent(in)   :: X
      Type(Input_data), Intent(in) :: Input_rec

      Real (kind=10) :: A_u, A_v, alpha_u, alpha_v, beta_u, beta_v
      Real (kind=10) :: amp_u, amp_v, amp_u_sqr, amp_v_sqr, Nu, q, w_u_sqr, w_v_sqr
      Real (kind=10) :: gamma_1, gamma_2, gamma_3, Xpos1, Xpos2
      Real (kind=10) :: Q_2, Q_4, Q_5, A, A_sqr, B, B_sqr, w_u, w_v
      Real (kind=10) :: rho, rho_sqr, beta_u_sqr, beta_v_sqr
      Real (kind=10) :: work_1, work_2, work_3, work_4, work_5, work_6, work_7
```

```
      Real (kind=10) :: wk_1c, wk_2c, wk_3c, wk_4c, wk_5c, wk_6c
      Real (kind=10) :: NLS_sech_approx, NLS_sech_tanh, W_xpos_1, W_xpos_2

      print*, 'Start Potential X ',X,'Input_rec',Input_rec,Xpos1,Xpos2

      A_u = Input_rec%Diff_1
      A_v = Input_rec%Diff_2
      A   = Input_rec%AAAA
      B   = Input_rec%BBBB
      amp_u = Input_rec%Amp_1
      amp_v = Input_rec%Amp_2
      w_u = abs(Input_rec%Width_1)
      w_v = abs(Input_rec%Width_2)

      alpha_u = Input_rec % alpha1
      alpha_v = Input_rec % alpha2
      beta_u  = Input_rec % beta1
      beta_v  = Input_rec % beta2
!      beta_v  = 11.0_10

      Nu   = Input_rec%Nu
      q    = Input_rec%qs

      A_sqr = A * A
      B_sqr = B * B
      amp_u_sqr = amp_u * amp_u
      amp_v_sqr = amp_v * amp_v
      w_u_sqr = w_u * w_u
      W_v_sqr = w_v * w_v
      beta_u_sqr = beta_u * beta_u
      beta_v_sqr = beta_v * beta_v

      print*, 'Potential 0 beta_v ', beta_v, 'beta_v_sqr ', beta_v_sqr

      rho = X

      W_xpos_1 = Xpos1
      W_xpos_2 = Xpos2

!      wk_1c  = NLS_sech_approx(W_xpos_1, W_xpos_2, w_u, beta_u, Input_rec)
!      wk_1c  = NLS_sech_approx(W_xpos_1, W_xpos_1, w_u, beta_u, Input_rec)
!      work_1 = 4.0_10 * A_u * amp_u_sqr * alpha_u * wk_1c
      wk_2c  = NLS_sech_approx(W_xpos_1, W_xpos_2, w_u, beta_v, Input_rec)
      work_2 = 4.0_10 * A_u * amp_u_sqr * alpha_v * wk_2c
      wk_3c  = NLS_sech_approx(W_xpos_1, W_xpos_2, beta_u, w_v, Input_rec)
      work_3 = 4.0_10 * A_v * amp_v_sqr * alpha_u * wk_3c
!      wk_4c  = NLS_sech_approx(W_xpos_1, W_xpos_2, w_v, beta_v, Input_rec)
!      wk_4c  = NLS_sech_approx(W_xpos_2, W_xpos_2, w_v, beta_v, Input_rec)
!      work_4 = 4.0_10 * A_v * amp_v_sqr * alpha_v * wk_4c
      wk_5c  = NLS_sech_approx(W_xpos_1, W_xpos_2, beta_u, beta_v, Input_rec)
      work_5 = -4.0_10 * q * alpha_u * alpha_v * wk_5c
      wk_6c  = NLS_sech_tanh(W_xpos_1, W_xpos_2, beta_u, beta_v, Input_rec)
!      work_7 = 16.0_10 * alpha_u * alpha_u / (15.0_10 * beta_u)
!      work_7 = work_7 + 2.0_10 * alpha_u * alpha_v * wk_6c + 16.0_10 * alpha_v * alpha_v / (15.0_10 * beta_v)
      work_6 = -2.0_10 * Nu * alpha_u * alpha_v * wk_6c

      print*, 'Pot X1', Xpos1, 'X2', Xpos2, 'B_u', beta_u, 'B_v', beta_v, 'work_7', work_7
      print*, 'Pot w2c', wk_2c, 'w3c', wk_3c, 'w5c', wk_5c, 'w6c', wk_6c
      print*, 'Pot w2', work_2, 'w3', work_3, 'w5', work_5, 'w6', work_6
      potential = (work_2 + work_3 + work_5 + work_6)

      Return
      End
!*
!**********************************************************************
!*
      Real (kind=10) Function ODE_func(W_mass, W_energy, X, Input_rec, iter, v2, Xpos1, Xpos2)
      implicit none
!
!     This is the function that is solved to calculate X, the difference between Xi_1 and Xi_2.
!

      Type Input_data
         Integer :: N_x, N_y, N_steps, NS_x, NS_y, Imid, IDebug, Ncont, Reg_w1
```

```
         Real (kind=10) :: T_step, T_initial, T_final
         Real (kind=10) :: Space_x, Space_y
         Real (kind=10) :: X_min, X_max, Y_min, Y_max
         Real (kind=10) :: Amp_1, Amp_2, Width_1, Width_2
         Real (kind=10) :: X_d1, Y_d1, Vel_1, Target_x
         Real (kind=10) :: X_d2, Y_d2, Vel_2
         Real (kind=10) :: Diff_1, Diff_2, AAAA, BBBB
         Real (kind=10) :: Nu, Theta, qs, R_phase, EPS, Om
         Real (kind=10) :: N_dp1, N_dp2, Damp_1, Damp_2, SRF
         Real (kind=10) :: Phi, Pi
         Real (kind=10) :: alpha1, alpha2, beta1, beta2, grad, inter, w1, w2, Reg_amt
      End Type

      Real (kind=10), Intent(in)   :: X, W_mass, W_energy, v2
      Type(Input_data), Intent(in) :: Input_rec
      Integer, Intent(in)          :: iter
      Real (kind=10)               :: Potential, W_sign, W_vel, Xpos1, Xpos2
      Real (kind=10)               :: W_work, W_work1, W_work2
      Logical :: W_sign_changed
      Save W_sign_changed, W_sign

      If (iter .eq. 1) then
         W_sign_changed = .false.
         W_vel = Input_rec%Vel_1-v2
         if (W_vel .ge. 0.0_10) then
            W_sign = 1.0_10
         else
            W_sign = -1.0_10
         End if
      End if

      W_work1 = Potential(X, Input_rec, Xpos1, Xpos2)
      W_work2 = (W_energy-2.0_10 * W_work1)
      W_work = 2.0_10 / W_mass * W_work2
!      If (mod(iter,100) .eq. 0) then
!          Print*,'ODE W_work ', W_work, ' W_work1 ', W_work1, ' W_work2 ', W_work2, ' W_energy ', W_energy
!          Print*,'ODE W_mass ', W_mass, ' W_sign ', W_sign
!      End if
      if (W_work .lt. 0.0_10) then
         W_work = -W_work
         if (W_sign_changed .eqv. .false.) then
             W_sign_changed = .true.
             W_sign = -W_sign
         end if
      end if

      ODE_func = sqrt(W_work)*W_sign

      Return
      End
!*
!**********************************************************************
!*
      Real (kind=10) Function Predict(Xpos,Vel,iter,TarX,Xpos2,M_cross,M_Bnd)

implicit none
!
!     Predict Vel from Xpos using Lagrangian interpolation
!     An array is formed from Xpos and the Velocity (the angle
!     the laser beam is initially directed). The process is stopped
!     when Xpos is equal to the Target value of X.
!
      Real (kind=10) X_array(25), X_weight(25), Vel_array(25,25), Vel_vect(25)
      Real (kind=10) Xpos,Xpos2,Vel,TarX
      Real (kind=10) Max_cross,Min_Bnd
      Real (kind=10) Wrk1,Wrk2,Wrk3,Wrk4,Wrk5, W_num, W_den
      Real (kind=10) Diff_1,Diff_2,Diff_3,X_h,X_l,Vel_h,Vel_l
      Real (kind=10) Abs_diff_1,Abs_diff_2,Abs_diff_3, Sum_diff, PrimRad
      Integer iter, j, M_i, M_Cross, M_Bnd, k, Same_vel
      Save X_array,Vel_array,X_weight,M_i,Max_cross,Min_Bnd

! Write(6,*) 'Start Predict'

If (iter .eq. 1) then
```

```
    M_i=0
    Max_cross=  0.5D0
    Min_bnd  = -1.5D0
Endif

If (M_cross .eq. 1) then
        If (Vel .lt. Max_cross) then
    Max_cross =Vel
        Endif
if (Xpos .lt. Xpos2) then
    Wrk1=Vel+(Xpos2-Xpos)*1.5d-3
else
    Wrk1=Vel+1.0d-3
Endif
ElseIf (M_bnd .eq. 1) then
If (Vel .gt. Min_Bnd) then
    Min_Bnd=Vel
Endif
Wrk1=0.5d0*(Max_Cross+Min_Bnd)
Endif

! Write(6,*) ' Max Cross ',Max_Cross,' Min Bnd ',Min_Bnd,' Vel ', Vel
! Write(6,*) ' M_i ', M_i,'TarX',TarX,'Xpos',Xpos

        Vel_vect(iter)=Vel
If ((M_Cross .eq. 0) .and. (M_Bnd .eq. 0)) then
   M_i=M_i+1
   X_array(M_i)=Xpos
           X_weight(M_i)=1.0_10 / abs(Xpos-TarX)
   Vel_array(M_i,1)=Vel
   if (M_i .eq. 1) then
              Wrk1=Vel-(TarX-Xpos)/30.0D0
              j=1
           else
              Do j=1,M_i-1
         Wrk2=X_array(M_i)
         Wrk3=X_array(M_i-j)
                 Wrk4=Vel_array(M_i-1,j)
                 Wrk5=Vel_array(M_i,j)
         Wrk1=((TarX-Wrk2)*Wrk4-(TarX-Wrk3)*Wrk5)/(Wrk3-Wrk2)
                 Wrk1=PrimRad(Wrk1)
!               Print*,'Pred wk1',wrk1,' wk2',wrk2,' wrk3',wrk3,
!     +                ' wrk4',wrk4,' wrk5',wrk5
        WRITE (6,1305) iter,Wrk1,Wrk2,Wrk3,Wrk4,Wrk5
                 Vel_array(M_i,j+1)=Wrk1
              Enddo
           Endif
           If (Wrk1 .gt. Max_Cross) then
      Wrk1=Max_Cross
           Endif
   If (Wrk1 .lt. Min_Bnd) then
      Wrk1=Min_Bnd
           Endif
         else
           If (M_cross .eq. 1) then
              Wrk1 = Vel-0.1DO*iter/(iter+1.0D0)
              If (Vel .LT. Max_Cross) then
         Max_Cross=Vel
              Endif
              If (Wrk1 .gt. Max_Cross) then
         Wrk1=Max_Cross
              Endif
           Endif
           If (M_bnd .eq. 1) then
              Wrk1 = Vel+0.1DO*iter/(iter+1.0D0)
              If (Vel .GT. Min_Bnd) then
         Min_Bnd=Vel
              Endif
              If (Wrk1 .lt. Min_Bnd) then
         Wrk1=Min_Bnd
              Endif
           Endif
Endif
!
```

```
!       The next piece of code calculates the weighted average of all the values
!       that did not cross or hit the boundary. The weight is 1/abs(Xpos-Tarx).
!       The weighted value is (Sum(vel * weight)+latest estimate)/(Sum(weight)+1).
!       So the latest estimate is given a weight of 1.0.
!



!
!       Check to see if the latest estimate is the same as a previous estimate
!
        Same_vel = 0
        Do k=1, iter
           Diff_1 = Vel_vect(k)
           Diff_2 = abs(Wrk1 - Diff_1)
           If (Diff_2 .lt. 1.0D-10) then
              Same_vel = 1
              Exit
           End if
        End do
        X_h   = -50.0D0
        X_l   =  50.0D0
        Vel_h = -1.5D0
        Vel_l =  1.5D0
        If (Same_vel .eq. 1) then
           Print*,'Pred same value 1',Wrk1,iter
           W_num = 0.0_10
           W_den = 0.0_10
!           If (M_i .gt. 3) then
             Do j=1,M_i
                W_num = W_num + Vel_array(j,1) * X_weight(j)
                W_den = W_den + X_weight(j)
             Enddo
!           Endif
              Print*,'Pred wk1',wrk1,' num',W_num,' den',W_den
           Wrk1 = (W_num + Wrk1) / (W_den + 1.0_10)
               Print*,'Pred wk1',wrk1
!           Do k=1, M_i
!              Diff_1 = TarX - X_array(k)
!              Diff_2 = TarX - X_h
!              Diff_3 = TarX - X_l
!              Abs_diff_1 = abs(Diff_1)
!              Abs_diff_2 = abs(Diff_2)
!              Abs_diff_3 = abs(Diff_3)
!              If (Diff_1 .lt. 0.0D0) then
!                 If (Abs_diff_1 .lt. Abs_diff_3) then
!                     X_l = X_array(k)
!                     Vel_l = Vel_vect(k)
!                 Endif
!              else
!                 If (Abs_diff_1 .lt. Abs_diff_2) then
!                     X_h = X_array(k)
!                     Vel_h = Vel_vect(k)
!                 Endif
!              Endif
!           Enddo
!           Abs_diff_2 = abs(TarX - X_h)
!           Abs_diff_3 = abs(TarX - X_l)
!           Sum_diff   = Abs_diff_2 + Abs_diff_3
!           Wrk1=(Abs_diff_2*Vel_h+Abs_diff_3*Vel_l)/Sum_diff
           Print*,'Pred same value 2',Wrk1,X_h,X_l
        End if

Predict=Wrk1

! Write(6,*) 'End Predict'

1305    Format (I4,' Wrk1',F12.7,' Wrk2',F12.7,' Wrk3',F12.7,' Wrk4',F12.7,' Wrk5',F12.7)
        Return
        End

!*
!**********************************************************************
!*
      Real (kind=10) Function NLS_sech_approx(W_X1, W_X2, W_W1, W_W2, Input_rec)
```

```
      implicit none
!
!     This function calculates the value of the integral of sech((x-x1)/w1)^2sech((x-x2)/w2)^2.
!

      Type Input_data
         Integer :: N_x, N_y, N_steps, NS_x, NS_y, Imid, IDebug, Ncont, Reg_w1
         Real (kind=10) :: T_step, T_initial, T_final
         Real (kind=10) :: Space_x, Space_y
         Real (kind=10) :: X_min, X_max, Y_min, Y_max
         Real (kind=10) :: Amp_1, Amp_2, Width_1, Width_2
         Real (kind=10) :: X_d1, Y_d1, Vel_1, Target_x
         Real (kind=10) :: X_d2, Y_d2, Vel_2
         Real (kind=10) :: Diff_1, Diff_2, AAAA, BBBB
         Real (kind=10) :: Nu, Theta, qs, R_phase, EPS, Om
         Real (kind=10) :: N_dp1, N_dp2, Damp_1, Damp_2, SRF
         Real (kind=10) :: Phi, Pi
         Real (kind=10) :: alpha1, alpha2, beta1, beta2, grad, inter, w1, w2, Reg_amt
      End Type

      Real (kind=10), Intent(in)   :: W_X1, W_X2, W_W1, W_W2
      Type(Input_data), Intent(inout) :: Input_rec
      Real (kind=10) :: W_work_1, W_work_2, W_work_3, W_work_4, W_work_5, W_work_6
      Real (kind=10) :: W_Mod_X, W_exp_w1, W_exp_w2
      Real (kind=10), Parameter :: C1 = -1.896324263_10, C2 = 1.010145_10, C3 = -0.015580_10
      Real (kind=10) :: W_w1_sqr,W_w2_sqr,W_w1_w2_sqr, K1, K2

      W_work_1 = 0.0_10
      W_work_2 = 0.0_10
      W_work_3 = 0.0_10
      W_work_4 = 0.0_10
      W_Mod_X = abs(W_X2 - W_X1)

!
!     When W1 = W2
!
      W_exp_w1 = exp(-2.0_10 * W_Mod_X / W_W1)
      W_w1_sqr = W_w1 * W_w1
      W_w2_sqr = W_w2 * W_w2
      W_w1_w2_sqr = (W_w1+W_w2)*(W_w1+W_w2)

!      Print*,'Sech_approx Mod x ',W_mod_x,'w1 ',W_w1,'w2 ',W_w2

      If (abs(W_W1 - W_W2) .le. 1.0D-5) then
         W_work_1 = (W_w1*0.5_10+W_mod_x)*W_w1_sqr*W_w1*0.125_10
         W_work_4 = W_work_1*W_exp_w1
      else
 W_exp_w2 = exp(-2.0_10 * W_Mod_X / W_W2)
         W_work_1 = W_w1_sqr*W_w2_sqr
         W_work_2 = (W_w1+W_w2)*(W_w1_sqr-W_w2_sqr)
         W_work_5 = (1.0_10/W_w1+1.0_10/W_w2)
         W_work_5 = W_work_5 * W_work_5
         W_work_3 = W_work_1/W_work_2*(W_w1*W_exp_w1-W_w2*W_exp_w2)/W_work_5
         Input_rec%w1 = W_w1
         Input_rec%w2 = W_w2
         call Regression(Input_Rec)
         K1 = Input_rec%Reg_amt
         W_work_6 = 1.0_10-C2-C3*W_mod_X*0.5_10
         If (W_work_6 .lt. 1.0D-5) then
            W_work_4 = W_work_1/(0.229613_10*W_w1_sqr+0.447971*W_w2_sqr)
         else
            K2 = log(1.0_10-C2-C3*W_mod_X*0.5_10)/C1
            W_work_4 = W_work_3 / (K1 * K2)
         end if
      End if

!      Print*, 'Sech_approx work_1',W_work_1, 'work_2', W_work_2, 'work_3', W_work_3, 'work_4', W_work_4, 'work_5', W_work_5
!      Print*, 'Sech_approx Exp1  ',W_exp_w1, 'Exp2', W_exp_w2, 'K1 ', K1, 'K2', K2

      NLS_sech_approx = W_work_4

      Return
      End
!*
```

```
!***********************************************************************
!*
      SUBROUTINE Regression(Input_rec)
      implicit none
      Type Input_data
         Integer :: N_x, N_y, N_steps, NS_x, NS_y, Imid, IDebug, Ncont, Reg_w1
         Real (kind=10) :: T_step, T_initial, T_final
         Real (kind=10) :: Space_x, Space_y
         Real (kind=10) :: X_min, X_max, Y_min, Y_max
         Real (kind=10) :: Amp_1, Amp_2, Width_1, Width_2
         Real (kind=10) :: X_d1, Y_d1, Vel_1, Target_x
         Real (kind=10) :: X_d2, Y_d2, Vel_2
         Real (kind=10) :: Diff_1, Diff_2, AAAA, BBBB
         Real (kind=10) :: Nu, Theta, qs, R_phase, EPS, Om
         Real (kind=10) :: N_dp1, N_dp2, Damp_1, Damp_2, SRF
         Real (kind=10) :: Phi, Pi
         Real (kind=10) :: alpha1, alpha2, beta1, beta2, grad, inter, w1, w2, Reg_amt
      End Type
      Type(Input_data), intent(inout) :: Input_rec

      Integer :: i,j,k
      Real (kind=10) :: W_w1,W_w2,W_grad,W_inter,int_w1, int_w2 ,W_frac , W_reg_w

      Real :: Grad(0:21), Inter(0:21)

      Data Grad(0:5)   /0.0_10, .003308_10, .013054_10, .026147_10, .040606_10, .056071_10/
      Data Grad(6:10)  /.072821_10, .091090_10, .110909_10, .132154_10, .154626_10/
      Data Grad(11:15) /.178107_10, .202394_10, .227305_10, .252689_10, .278420_10/
      Data Grad(16:21) /.278420_10, .003845_10, .0039755_10, .013791_10, .012588_10, .012588_10/

      Data Inter(0:5)   /0.0_10, .011065_10, .004323_10, -.018982_10, -.046262_10, -.073026_10/
      Data Inter(6:10)  /-.100306_10, -.130175_10, -.163980_10, -.202174_10, -.244603_10/
      Data Inter(11:15) /-.290812_10, -.340240_10, -.392331_10, -.446582_10, -.502561_10/
      Data Inter(16:21) /-.502561_10, -.001253_10, .005705_10, -.005133_10, .001071_10, .001071_10/

      W_w1 = Input_rec%w1
      W_w2 = Input_rec%w2
      i = int(W_w1)
      int_w1 = i * 1.0_10
      j = int(W_w2)
      int_w2 = j * 1.0_10
      if (i .gt. 15) then
         i = 15
      end if
      if (i .lt. 0) then
         i = 0
      end if
      if (j .gt. 15) then
         j = 15
      end if
      if (j .lt. 0) then
         j = 0
      end if
      If (j .le. 2) then
         Input_rec%Reg_w1 = 1
         W_reg_w = W_w1
         if (W_w2 .le. 1.0_10) then
            if (W_w1 .le. 1.0_10) then
               k = 17
               W_frac = W_w1 - int_w1
            else
               k = 18
               W_frac = W_w1 - int_w1
            end if
         else
            if (W_w1 .le. 1.0_10) then
               k = 19
               W_frac = W_w1 - int_w1
            else
               k = 20
               W_frac = W_w1 - int_w1
            end if
         end if
      else
```

```
         Input_rec%Reg_w1 = 0
         W_reg_w = W_w2
         k = i
         W_frac = W_w2 - int_w2
      End if

      W_Grad = Grad(k) * (1.0_10 - W_frac) + Grad(k+1) * W_frac
      W_inter = Inter(k) * (1.0_10 - W_frac) + Inter(k+1) * W_frac

      Input_rec%grad = W_Grad
      Input_rec%inter = W_inter
      Input_rec%Reg_amt = W_inter + W_Grad * W_reg_w

      End subroutine
!*
!**********************************************************************
!*
      Real (kind=10) Function NLS_sech_tanh(W_X1, W_X2, W_W1, W_W2, Input_rec)
      implicit none
!
!     This function calculates the value of the integral of tanh((x-x1)/w1)*sech((x-x1)/w1)^2*tanh((x-x2)/w2)*sech((x-x2)/w2)^2.
!

      Type Input_data
         Integer :: N_x, N_y, N_steps, NS_x, NS_y, Imid, IDebug, Ncont, Reg_w1
         Real (kind=10) :: T_step, T_initial, T_final
         Real (kind=10) :: Space_x, Space_y
         Real (kind=10) :: X_min, X_max, Y_min, Y_max
         Real (kind=10) :: Amp_1, Amp_2, Width_1, Width_2
         Real (kind=10) :: X_d1, Y_d1, Vel_1, Target_x
         Real (kind=10) :: X_d2, Y_d2, Vel_2
         Real (kind=10) :: Diff_1, Diff_2, AAAA, BBBB
         Real (kind=10) :: Nu, Theta, qs, R_phase, EPS, Om
         Real (kind=10) :: N_dp1, N_dp2, Damp_1, Damp_2, SRF
         Real (kind=10) :: Phi, Pi
         Real (kind=10) :: alpha1, alpha2, beta1, beta2, grad, inter, w1, w2, Reg_amt
      End Type

      Real (kind=10), Intent(in)   :: W_X1, W_X2, W_W1, W_W2
      Type(Input_data), Intent(in) :: Input_rec
      Real (kind=10) :: W_work1, W_work2, W_work3, W_work4, W_work5
      Real (kind=10) :: NLS_sech_approx
      Real (kind=10), parameter :: W_dev=.005_10

      W_work1 = NLS_sech_approx(W_X1+W_dev, W_X2+W_dev, W_W1, W_W2, Input_rec)
      W_work2 = NLS_sech_approx(W_X1-W_dev, W_X2+W_dev, W_W1, W_W2, Input_rec)
      W_work3 = NLS_sech_approx(W_X1+W_dev, W_X2-W_dev, W_W1, W_W2, Input_rec)
      W_work4 = NLS_sech_approx(W_X1-W_dev, W_X2-W_dev, W_W1, W_W2, Input_rec)
      W_work5 = (W_work1-W_work2-W_work3+W_work4)/(4.0_10*W_dev*W_dev)

!      Print*, 'Sech_tanh work_1',W_work1, 'work_2', W_work2, 'work_3', W_work3, 'work_4', W_work4
!      Print*, 'Sech_tanh W_x1',W_X1, 'W_x2', W_X2, 'W_w1', W_W1, 'W_w2', W_W2, 'work_5', W_work5

      NLS_sech_tanh = W_work5

      Return
      End
!
!     This function calculates the value of the width and amplitude of the director for each laser beam.
!
      Subroutine BetaFind(Input_rec)
      implicit none

      Type Input_data
         Integer :: N_x, N_y, N_steps, NS_x, NS_y, Imid, IDebug, Ncont, Reg_w1
         Real (kind=10) :: T_step, T_initial, T_final
         Real (kind=10) :: Space_x, Space_y
         Real (kind=10) :: X_min, X_max, Y_min, Y_max
         Real (kind=10) :: Amp_1, Amp_2, Width_1, Width_2
         Real (kind=10) :: X_d1, Y_d1, Vel_1, Target_x
         Real (kind=10) :: X_d2, Y_d2, Vel_2
         Real (kind=10) :: Diff_1, Diff_2, AAAA, BBBB
         Real (kind=10) :: Nu, Theta, qs, R_phase, EPS, Om
         Real (kind=10) :: N_dp1, N_dp2, Damp_1, Damp_2, SRF
```

```
        Real (kind=10) :: Phi, Pi
        Real (kind=10) :: alpha1, alpha2, beta1, beta2, grad, inter, w1, w2, Reg_amt
      End Type

      Integer :: iter, iter2, Min_x
      Real (kind=10) :: Func_1, Func_2, Func_3, Func_4
      Real (kind=10) :: Grad_F1_alpha_u, Grad_F1_alpha_v, Grad_F1_beta_u, Grad_F1_beta_v
      Real (kind=10) :: Grad_F2_alpha_u, Grad_F2_alpha_v, Grad_F2_beta_u, Grad_F2_beta_v
      Real (kind=10) :: Grad_F3_alpha_u, Grad_F3_alpha_v, Grad_F3_beta_u, Grad_F3_beta_v
      Real (kind=10) :: Grad_F4_alpha_u, Grad_F4_alpha_v, Grad_F4_beta_u, Grad_F4_beta_v
      Type(Input_data) :: Input_rec
      Integer :: iterations
      Real (kind=10) :: W_a_1, W_a_2, W_b_1, W_b_2, W_a0, W_a2, W_a3, W_a4, W_a
      Real (kind=10) :: W_x0_1, W_x0_2, W_x0_3, W_x0_4
      Real (kind=10) :: W_x1_1, W_x1_2, W_x1_3, W_x1_4
      Real (kind=10) :: W_x_1, W_x_2, W_x_3, W_x_4
      Real (kind=10) :: x1,x2,x3,x4
      Real (kind=10) :: W_z_1, W_z_2, W_z_3, W_z_4, W_mod_z
      Real (kind=10) :: W_f_1, W_f_2, W_f_3, W_f_4
      Real (kind=10) :: W_tol= 0.00005_10
      Real (kind=10) :: W_gf1_1, W_gf1_2, W_gf1_3, W_gf1_4
      Real (kind=10) :: W_gf2_1, W_gf2_2, W_gf2_3, W_gf2_4
      Real (kind=10) :: W_gf3_1, W_gf3_2, W_gf3_3, W_gf3_4
      Real (kind=10) :: W_gf4_1, W_gf4_2, W_gf4_3, W_gf4_4
      Real (kind=10) :: W_min_1, W_min_2, W_min_3, W_min_4, W_min
      Real (kind=10) :: W_h_1, W_h_2, W_h_3, W_abs

      W_a_1 = 1.0_10
      W_a_2 = 1.0_10
      W_b_1 = 10.0_10
      W_b_2 = 10.0_10

      W_x_1 = W_a_1
      W_x_2 = W_a_2
      W_x_3 = W_b_1
      W_x_4 = W_b_2

!      If (W_tol .gt. 0.5) then
      Do iter2 = 1, 10

      W_x0_1 = W_x_1
      W_x0_2 = W_x_2
      W_x0_3 = W_x_3
      W_x0_4 = W_x_4

!       Print*, 'Call functions X1 ', W_x0_1, ' X2 ', W_x0_2, ' X3 ', W_x0_3, ' X4 ', W_x0_4

      W_f_1 = Func_1(W_x0_1, W_x0_2, W_x0_3, W_x0_4, Input_rec)
      W_f_2 = Func_2(W_x0_1, W_x0_2, W_x0_3, W_x0_4, Input_rec)
      W_f_3 = Func_3(W_x0_1, W_x0_2, W_x0_3, W_x0_4, Input_rec)
      W_f_4 = Func_4(W_x0_1, W_x0_2, W_x0_3, W_x0_4, Input_rec)

      W_min_1 = W_f_1 * W_f_1 + W_f_2 * W_f_2 + W_f_3 * W_f_3 + W_f_4 * W_f_4

!       Print*, 'Call grads W_min_1 ', W_min_1, ' F1 ', W_f_1, ' F2 ', W_f_2, ' F3 ', W_f_3, ' F4 ', W_f_4

      W_gf1_1 = Grad_F1_alpha_u(W_x0_1, W_x0_2, W_x0_3, W_x0_4, Input_rec)
      W_gf1_2 = Grad_F1_alpha_v(W_x0_1, W_x0_2, W_x0_3, W_x0_4, Input_rec)
      W_gf1_3 = Grad_F1_beta_u(W_x0_1, W_x0_2, W_x0_3, W_x0_4, Input_rec)
      W_gf1_4 = Grad_F1_beta_v(W_x0_1, W_x0_2, W_x0_3, W_x0_4, Input_rec)

!       Print*, 'Grad F1 1 ', W_gf1_1, 'Grad F1 2 ', W_gf1_2, 'Grad F1 3 ', W_gf1_3, 'Grad F1 4 ', W_gf1_4

      W_gf2_1 = Grad_F2_alpha_u(W_x0_1, W_x0_2, W_x0_3, W_x0_4, Input_rec)
      W_gf2_2 = Grad_F2_alpha_v(W_x0_1, W_x0_2, W_x0_3, W_x0_4, Input_rec)
      W_gf2_3 = Grad_F2_beta_u(W_x0_1, W_x0_2, W_x0_3, W_x0_4, Input_rec)
      W_gf2_4 = Grad_F2_beta_v(W_x0_1, W_x0_2, W_x0_3, W_x0_4, Input_rec)

!       Print*, 'Grad F2 1 ', W_gf2_1, 'Grad F2 2 ', W_gf2_2, 'Grad F2 3 ', W_gf2_3, 'Grad F2 4 ', W_gf2_4

      W_gf3_1 = Grad_F3_alpha_u(W_x0_1, W_x0_2, W_x0_3, W_x0_4, Input_rec)
      W_gf3_2 = Grad_F3_alpha_v(W_x0_1, W_x0_2, W_x0_3, W_x0_4, Input_rec)
      W_gf3_3 = Grad_F3_beta_u(W_x0_1, W_x0_2, W_x0_3, W_x0_4, Input_rec)
      W_gf3_4 = Grad_F3_beta_v(W_x0_1, W_x0_2, W_x0_3, W_x0_4, Input_rec)
```

```
!       Print*, 'Grad F3 1 ', W_gf3_1, 'Grad F3 2 ', W_gf3_2, 'Grad F3 3 ', W_gf3_3, 'Grad F3 4 ', W_gf3_4

      W_gf4_1 = Grad_F4_alpha_u(W_x0_1, W_x0_2, W_x0_3, W_x0_4, Input_rec)
      W_gf4_2 = Grad_F4_alpha_v(W_x0_1, W_x0_2, W_x0_3, W_x0_4, Input_rec)
      W_gf4_3 = Grad_F4_beta_u(W_x0_1, W_x0_2, W_x0_3, W_x0_4, Input_rec)
      W_gf4_4 = Grad_F4_beta_v(W_x0_1, W_x0_2, W_x0_3, W_x0_4, Input_rec)

!       Print*, 'Grad F4 1 ', W_gf4_1, 'Grad F4 2 ', W_gf4_2, 'Grad F4 3 ', W_gf4_3, 'Grad F4 4 ', W_gf4_4

      W_z_1 = 2.0_10 * (W_f_1 * W_gf1_1 + W_f_2 * W_gf2_1 + W_f_3 * W_gf3_1 +W_f_4 * W_gf4_1)
      W_z_2 = 2.0_10 * (W_f_1 * W_gf1_2 + W_f_2 * W_gf2_2 + W_f_3 * W_gf3_2 +W_f_4 * W_gf4_2)
      W_z_3 = 2.0_10 * (W_f_1 * W_gf1_3 + W_f_2 * W_gf2_3 + W_f_3 * W_gf3_3 +W_f_4 * W_gf4_3)
      W_z_4 = 2.0_10 * (W_f_1 * W_gf1_4 + W_f_2 * W_gf2_4 + W_f_3 * W_gf3_4 +W_f_4 * W_gf4_4)
      W_mod_z = sqrt(W_z_1 * W_z_1 + W_z_2 * W_z_2 + W_z_3 * W_z_3 + W_z_4 * W_z_4)

!       Print*, 'z 1 ', W_z_1, 'z 2 ', W_z_2, 'z 3 ', W_z_3, 'z 4 ', W_z_4, ' Mod z ', W_mod_z

      if (W_mod_z .ne. 0.0_10) then
 W_a3 = 1.0_10
 Do iter = 1, 10
    W_x1_1 = W_x0_1 - W_z_1 / W_mod_z * W_a3
    W_x1_2 = W_x0_2 - W_z_2 / W_mod_z * W_a3
    W_x1_3 = W_x0_3 - W_z_3 / W_mod_z * W_a3
    W_x1_4 = W_x0_4 - W_z_4 / W_mod_z * W_a3

    W_f_1 = Func_1(W_x1_1, W_x1_2, W_x1_3, W_x1_4, Input_rec)
    W_f_2 = Func_2(W_x1_1, W_x1_2, W_x1_3, W_x1_4, Input_rec)
    W_f_3 = Func_3(W_x1_1, W_x1_2, W_x1_3, W_x1_4, Input_rec)
    W_f_4 = Func_4(W_x1_1, W_x1_2, W_x1_3, W_x1_4, Input_rec)

    W_min_3 = W_f_1 * W_f_1 + W_f_2 * W_f_2 + W_f_3 * W_f_3 + W_f_4 * W_f_4

!  Print *, 'W_min_3 ', W_min_3, ' alpha u ', W_x1_1, ' alpha v ', W_x1_2, ' beta u ', W_x1_3, ' beta v ', W_x1_4

    If (W_min_3 .ge. W_min_1) then
       W_a3 = W_a3 * 0.5_10
    else
       exit
    End if
 End do

 W_a2 = W_a3 * 0.5_10
 W_x1_1 = W_x0_1 - W_z_1 / W_mod_z * W_a2
 W_x1_2 = W_x0_2 - W_z_2 / W_mod_z * W_a2
 W_x1_3 = W_x0_3 - W_z_3 / W_mod_z * W_a2
 W_x1_4 = W_x0_4 - W_z_4 / W_mod_z * W_a2

 W_f_1 = Func_1(W_x1_1, W_x1_2, W_x1_3, W_x1_4, Input_rec)
 W_f_2 = Func_2(W_x1_1, W_x1_2, W_x1_3, W_x1_4, Input_rec)
 W_f_3 = Func_3(W_x1_1, W_x1_2, W_x1_3, W_x1_4, Input_rec)
 W_f_4 = Func_4(W_x1_1, W_x1_2, W_x1_3, W_x1_4, Input_rec)

 W_min_2 = W_f_1 * W_f_1 + W_f_2 * W_f_2 + W_f_3 * W_f_3 + W_f_4 * W_f_4

!  Print *, 'W_min_2 ', W_min_2, ' alpha u ', W_x1_1, ' alpha v ', W_x1_2, ' beta u ', W_x1_3, ' beta v ', W_x1_4

 W_h_1 = (W_min_2 - W_min_1) / W_a2
 W_h_2 = (W_min_3 - W_min_2) / (W_a3 - W_a2)
 W_h_3 = (W_h_2 - W_h_1) / W_a3

 W_a4 = 0.5_10 * (W_a2 - W_h_1 / W_h_3)

 W_x1_1 = W_x0_1 - W_z_1 / W_mod_z * W_a4
 W_x1_2 = W_x0_2 - W_z_2 / W_mod_z * W_a4
 W_x1_3 = W_x0_3 - W_z_3 / W_mod_z * W_a4
 W_x1_4 = W_x0_4 - W_z_4 / W_mod_z * W_a4

 W_f_1 = Func_1(W_x1_1, W_x1_2, W_x1_3, W_x1_4, Input_rec)
 W_f_2 = Func_2(W_x1_1, W_x1_2, W_x1_3, W_x1_4, Input_rec)
 W_f_3 = Func_3(W_x1_1, W_x1_2, W_x1_3, W_x1_4, Input_rec)
 W_f_4 = Func_4(W_x1_1, W_x1_2, W_x1_3, W_x1_4, Input_rec)

 W_min_4 = W_f_1 * W_f_1 + W_f_2 * W_f_2 + W_f_3 * W_f_3 + W_f_4 * W_f_4
```

```
!  Print *,' alpha u ', W_x1_1, ' alpha v ', W_x1_2, ' beta u ', W_x1_3, ' beta v ', W_x1_4
!                 Print*,'W_f_1 ',W_f_1,'W_f_2 ',W_f_2,'W_f_3 ',W_f_3,'W_f_4 ',W_f_4, 'W_min_4 ', W_min_4

 If (W_min_1 .lt. W_min_3) then
    W_min = W_min_1
    Min_x = 1
 else
    W_min = W_min_3
    Min_x = 3
 Endif
 If (W_min_4 .lt. W_min) then
    W_min = W_min_4
    Min_x = 4
 Endif
 If (Min_x .eq. 1) then
    W_a = 0.0_10
 elseif (Min_x .eq. 3) then
    W_a = W_a3
 else
    W_a = W_a4
 Endif

 W_x_1 = W_x0_1 - W_z_1 / W_mod_z * W_a
 W_x_2 = W_x0_2 - W_z_2 / W_mod_z * W_a
 W_x_3 = W_x0_3 - W_z_3 / W_mod_z * W_a
 W_x_4 = W_x0_4 - W_z_4 / W_mod_z * W_a
 W_abs = abs(W_min)

!  Print *, 'G ', W_min, ' (', W_x_1, W_x_2, W_x_3, W_x_4, ') Error ', W_abs
!                 Print*
 If (W_abs .lt. W_tol) then
    exit
 Endif
      End if

      End do
!      End if

      x1=W_x_1
      x2=W_x_2
      x3=W_x_3
      x4=W_x_4

!  Print *, 'G ', W_min, ' (', W_x_1, W_x_2, W_x_3, W_x_4, ') Error ', W_abs
!                Print*
      If (W_tol .gt. 0.5) then
      Do iter=1, 11
                x1=W_x_1+0.0001_10*(iter-5)
 W_f_1 = Func_1(x1,x2,x3,x4,Input_rec)
 W_f_2 = Func_2(x1,x2,x3,x4,Input_rec)
 W_f_3 = Func_3(x1,x2,x3,x4,Input_rec)
 W_f_4 = Func_4(x1,x2,x3,x4,Input_rec)

 W_min_4 = W_f_1 * W_f_1 + W_f_2 * W_f_2 + W_f_3 * W_f_3 + W_f_4 * W_f_4

!  Print *,' x1 ', x1, ' x2 ', x2, ' x3 ', x3, ' x4 ', x4
!                 Print*,'W_f_1 ',W_f_1,'W_f_2 ',W_f_2,'W_f_3 ',W_f_3,'W_f_4 ',W_f_4, 'W_min_4 ', W_min_4
!                 Print*
      End do

      x1=W_x_1
      x2=W_x_2
      x3=W_x_3
      x4=W_x_4

      Do iter=1, 11
                x2=W_x_2+0.0001_10*(iter-5)
 W_f_1 = Func_1(x1,x2,x3,x4,Input_rec)
 W_f_2 = Func_2(x1,x2,x3,x4,Input_rec)
 W_f_3 = Func_3(x1,x2,x3,x4,Input_rec)
 W_f_4 = Func_4(x1,x2,x3,x4,Input_rec)

 W_min_4 = W_f_1 * W_f_1 + W_f_2 * W_f_2 + W_f_3 * W_f_3 + W_f_4 * W_f_4
```

```
!  Print *,' x1 ', x1, ' x2 ', x2, ' x3 ', x3, ' x4 ', x4
!                 Print*,'W_f_1 ',W_f_1,'W_f_2 ',W_f_2,'W_f_3 ',W_f_3,'W_f_4 ',W_f_4, 'W_min_4 ', W_min_4
!                 Print*
      End do

      x1=W_x_1
      x2=W_x_2
      x3=W_x_3
      x4=W_x_4

      Do iter=1, 11
                 x3=W_x_3+0.0001_10*(iter-5)
 W_f_1 = Func_1(x1,x2,x3,x4,Input_rec)
 W_f_2 = Func_2(x1,x2,x3,x4,Input_rec)
 W_f_3 = Func_3(x1,x2,x3,x4,Input_rec)
 W_f_4 = Func_4(x1,x2,x3,x4,Input_rec)

 W_min_4 = W_f_1 * W_f_1 + W_f_2 * W_f_2 + W_f_3 * W_f_3 + W_f_4 * W_f_4

!  Print *,' x1 ', x1, ' x2 ', x2, ' x3 ', x3, ' x4 ', x4
!                 Print*,'W_f_1 ',W_f_1,'W_f_2 ',W_f_2,'W_f_3 ',W_f_3,'W_f_4 ',W_f_4, 'W_min_4 ', W_min_4
!                 Print*
      End do
!                 Print*
      x1=W_x_1
      x2=W_x_2
      x3=W_x_3
      x4=W_x_4
!                 Print*, 'Start x4'

      Do iter=1, 11
                 x4=W_x_4+0.01_10*(iter-5)
 W_f_1 = Func_1(x1,x2,x3,x4,Input_rec)
 W_f_2 = Func_2(x1,x2,x3,x4,Input_rec)
 W_f_3 = Func_3(x1,x2,x3,x4,Input_rec)
 W_f_4 = Func_4(x1,x2,x3,x4,Input_rec)
                 W_gf2_4 = Grad_F2_beta_v(x1,x2,x3,x4,Input_rec)

 W_min_4 = W_f_1 * W_f_1 + W_f_2 * W_f_2 + W_f_3 * W_f_3 + W_f_4 * W_f_4

!  Print *,' x1 ', x1, ' x2 ', x2, ' x3 ', x3, ' x4 ', x4, ' W_gf2_4 ', W_gf2_4
!                 Print*,'W_f_1 ',W_f_1,'W_f_2 ',W_f_2,'W_f_3 ',W_f_3,'W_f_4 ',W_f_4, 'W_min_4 ', W_min_4
!                 Print*
      End do
      End if

!      W_x_1 = 0.18980457695489520107_10
!      W_x_2 = 0.18976206176105950816_10
!      W_x_3 = 11.975465610670008458_10
!      W_x_4 = 11.978154437922945834_10

      Call Newton(W_x_1, W_x_2, W_x_3, W_x_4, Input_rec)

      Input_rec % alpha1 = W_x_1
      Input_rec % alpha2 = W_x_2
      Input_rec % beta1  = W_x_3
      Input_rec % beta2  = W_x_4

 200  FORMAT (I10,14X,I6)
 220  FORMAT (F20.16)
 240  FORMAT (8X,I10,16X,A40)
 260  FORMAT (8X,F24.16,2X,A40)
 280  FORMAT (I6)

      End subroutine BetaFind
!
!     Use Newtons method and the intial values of x1, x2, x3, x4 to find the solutions
!     for the amplidude and width of the director for each of the beams.
!
        Subroutine Newton(x1, x2, x3, x4, State_rec)
        implicit none

!       Program to calculate a fixed point in a nonlinear system of equations
```

```
!       using Newton's method.

        Type Input_data
           Integer :: N_x, N_y, N_steps, NS_x, NS_y, Imid, IDebug, Ncont, Reg_w1
           Real (kind=10) :: T_step, T_initial, T_final
           Real (kind=10) :: Space_x, Space_y
           Real (kind=10) :: X_min, X_max, Y_min, Y_max
           Real (kind=10) :: Amp_1, Amp_2, Width_1, Width_2
           Real (kind=10) :: X_d1, Y_d1, Vel_1, Target_x
           Real (kind=10) :: X_d2, Y_d2, Vel_2
           Real (kind=10) :: Diff_1, Diff_2, AAAA, BBBB
           Real (kind=10) :: Nu, Theta, qs, R_phase, EPS, Om
           Real (kind=10) :: N_dp1, N_dp2, Damp_1, Damp_2, SRF
           Real (kind=10) :: Phi, Pi
         Real (kind=10) :: alpha1, alpha2, beta1, beta2, grad, inter, w1, w2, Reg_amt
        End Type
        Type(Input_data) :: State_rec
        Real (kind=10) :: x1, x2, x3, x4, x1_0, x2_0, x3_0, x4_0
        Real (kind=10) :: W_prev_x1,W_prev_x2,W_prev_x3,W_prev_x4
        Real (kind=10) :: y1, y2, y3, y4
        Real (kind=10) :: Func_1, Func_2, Func_3, Func_4
        Real (kind=10) :: Grad_F1_alpha_u, Grad_F1_alpha_v, Grad_F1_beta_u, Grad_F1_beta_v
        Real (kind=10) :: Grad_F2_alpha_u, Grad_F2_alpha_v, Grad_F2_beta_u, Grad_F2_beta_v
        Real (kind=10) :: Grad_F3_alpha_u, Grad_F3_alpha_v, Grad_F3_beta_u, Grad_F3_beta_v
        Real (kind=10) :: Grad_F4_alpha_u, Grad_F4_alpha_v, Grad_F4_beta_u, Grad_F4_beta_v
        Real (kind=10), Dimension(4, 4) :: Jacob
        Real (kind=10), Dimension(4)    :: F_vec
        Real (kind=10) :: W_coeff, W_mod_x, W_prev_x, W_Tol = 0.0000001_10
        Integer :: i, j, k, l

!        x1_0 = 0.2834_10
!        x2_0 = 0.2834_10
!        x3_0 = 10.01698_10
!        x4_0 = 10.02209_10

!        x1 = x1_0
!        x2 = x2_0
!        x3 = x3_0
!        x4 = x4_0

        W_prev_x1 = x1
        W_prev_x2 = x2
        W_prev_x3 = x3
        W_prev_x4 = x4

        do l=1, 10

!        Print*
!        Print*,'Newton ',x1,x2,x3,x4
!        Print*

        Jacob(1, 1) = Grad_F1_alpha_u(x1, x2, x3, x4, State_rec)
        Jacob(1, 2) = Grad_F1_alpha_v(x1, x2, x3, x4, State_rec)
        Jacob(1, 3) = Grad_F1_beta_u(x1, x2, x3, x4, State_rec)
        Jacob(1, 4) = Grad_F1_beta_v(x1, x2, x3, x4, State_rec)
        Jacob(2, 1) = Grad_F2_alpha_u(x1, x2, x3, x4, State_rec)
        Jacob(2, 2) = Grad_F2_alpha_v(x1, x2, x3, x4, State_rec)
        Jacob(2, 3) = Grad_F2_beta_u(x1, x2, x3, x4, State_rec)
        Jacob(2, 4) = Grad_F2_beta_v(x1, x2, x3, x4, State_rec)
        Jacob(3, 1) = Grad_F3_alpha_u(x1, x2, x3, x4, State_rec)
        Jacob(3, 2) = Grad_F3_alpha_v(x1, x2, x3, x4, State_rec)
        Jacob(3, 3) = Grad_F3_beta_u(x1, x2, x3, x4, State_rec)
        Jacob(3, 4) = Grad_F3_beta_v(x1, x2, x3, x4, State_rec)
        Jacob(4, 1) = Grad_F4_alpha_u(x1, x2, x3, x4, State_rec)
        Jacob(4, 2) = Grad_F4_alpha_v(x1, x2, x3, x4, State_rec)
        Jacob(4, 3) = Grad_F4_beta_u(x1, x2, x3, x4, State_rec)
        Jacob(4, 4) = Grad_F4_beta_v(x1, x2, x3, x4, State_rec)

        F_vec(1) = Func_1(x1, x2, x3, x4, State_rec)
        F_vec(2) = Func_2(x1, x2, x3, x4, State_rec)
        F_vec(3) = Func_3(x1, x2, x3, x4, State_rec)
        F_vec(4) = Func_4(x1, x2, x3, x4, State_rec)

!        Print*, 'Jacobian ', Jacob(1,1), Jacob(1,2), Jacob(1,3), Jacob(1,4), ' F1 ', F_vec(1)
```

```
!        Print*, '         ', Jacob(2,1), Jacob(2,2), Jacob(2,3), Jacob(2,4), ' F2 ', F_vec(2)
!        Print*, '         ', Jacob(3,1), Jacob(3,2), Jacob(3,3), Jacob(3,4), ' F3 ', F_vec(3)
!        Print*, '         ', Jacob(4,1), Jacob(4,2), Jacob(4,3), Jacob(4,4), ' F4 ', F_vec(4)
!        Print*, '         '

        Do k=1,3
   Do i=k+1,4
      W_coeff = Jacob(i,k) / Jacob(k,k)
      Do j=k,4
         Jacob(i,j) = Jacob(i,j) - Jacob(k,j) * W_coeff
      End do
      F_vec(i) = F_vec(i) - F_vec(k) * W_coeff
   End do
        End do

!        Print*, 'Jacobian ', Jacob(1,1), Jacob(1,2), Jacob(1,3), Jacob(1,4), ' F1 ', F_vec(1)
!        Print*, '         ', Jacob(2,1), Jacob(2,2), Jacob(2,3), Jacob(2,4), ' F2 ', F_vec(2)
!        Print*, '         ', Jacob(3,1), Jacob(3,2), Jacob(3,3), Jacob(3,4), ' F3 ', F_vec(3)
!        Print*, '         ', Jacob(4,1), Jacob(4,2), Jacob(4,3), Jacob(4,4), ' F4 ', F_vec(4)
!        Print*, '         '

        y4 = F_vec(4) / Jacob(4,4)
        y3 = (F_vec(3) - Jacob(3,4)*y4) / Jacob(3,3)
        y2 = (F_vec(2) - Jacob(2,4)*y4 - Jacob(2,3)*y3) / Jacob(2,2)
        y1 = (F_vec(1) - Jacob(1,4)*y4 - Jacob(1,3)*y3 - Jacob(1,2)*y2) / Jacob(1,1)

!        Print*, 'y ',y1, y2, y3, y4

        x1 = x1 - y1
        x2 = x2 - y2
        x3 = x3 - y3
        x4 = x4 - y4

!        W_mod_x = (W_prev_x1-x1)*(W_prev_x1-x1) + (W_prev_x2-x2)*(W_prev_x2-x2)
!        W_mod_x = W_mod_x + (W_prev_x3-x3)*(W_prev_x3-x3) + (W_prev_x4-x4)*(W_prev_x4-x4)
        W_mod_x = y1*y1+y2*y2+y3*y3+y4*y4
        W_mod_x = sqrt(W_mod_x)

!        Print*, 'x ', x1, x2, x3, x4, W_mod_x
!        Print*

        If (W_mod_x .gt. W_Tol) then
           W_prev_x1 = x1
           W_prev_x2 = x2
           W_prev_x3 = x3
           W_prev_x4 = x4
        else
           exit
        End if

        End do

        End subroutine Newton


      real (kind=10) Function Func_1(alpha_u, alpha_v, beta_u, beta_v, input_rec)
      implicit none

      Type Input_data
         Integer :: N_x, N_y, N_steps, NS_x, NS_y, Imid, IDebug, Ncont, Reg_w1
         Real (kind=10) :: T_step, T_initial, T_final
         Real (kind=10) :: Space_x, Space_y
         Real (kind=10) :: X_min, X_max, Y_min, Y_max
         Real (kind=10) :: Amp_1, Amp_2, Width_1, Width_2
         Real (kind=10) :: X_d1, Y_d1, Vel_1, Target_x
         Real (kind=10) :: X_d2, Y_d2, Vel_2
         Real (kind=10) :: Diff_1, Diff_2, AAAA, BBBB
         Real (kind=10) :: Nu, Theta, qs, R_phase, EPS, Om
         Real (kind=10) :: N_dp1, N_dp2, Damp_1, Damp_2, SRF
         Real (kind=10) :: Phi, Pi
         Real (kind=10) :: alpha1, alpha2, beta1, beta2, grad, inter, w1, w2, Reg_amt
      End Type

      Real (kind=10), intent(in) :: alpha_u, alpha_v, beta_u, beta_v
```

```
Type (Input_data), intent(in) :: input_rec
Real (kind=10) :: A_u, A_v, amp_u, amp_v, w_u, w_v, w_u_sqr, w_v_sqr, amp_u_sqr, amp_v_sqr
Real (kind=10) :: Const_A = 0.8429538023_10, Const_B = 1.177410023_10, Const_C = 1.1_10
Real (kind=10) :: I4 = 0.2954314537_10, I42 = 0.1090862907_10
Real (kind=10) :: A_sqr, B_sqr, C_sqr, Beta_u_sqr, Beta_v_sqr
Real (kind=10) :: Work_1, Work_2, Work_3, Work_4, Work_5, Work_6, Work_7, Work_8
Real (kind=10) :: Q1, Q2, Q3, Q4, Q5, rho, rho_sqr
Real (kind=10) :: gamma1, gamma2, gamma3, gamma4, gamma5

A_u = Input_rec%AAAA
A_v = Input_rec%BBBB
amp_u = Input_rec%Amp_1
amp_v = Input_rec%Amp_2
w_u = Input_rec%Width_1
w_v = Input_rec%Width_2
w_u_sqr = w_u * w_u
w_v_sqr = w_v * w_v
rho = Input_rec%X_d1 - Input_rec%X_d2
rho_sqr = rho * rho
A_sqr = Const_A * Const_A
B_sqr = Const_B * Const_B
C_sqr = Const_C * Const_C
Beta_u_sqr = beta_u * beta_u
Beta_v_sqr = beta_v * beta_v

Q1 = A_sqr * Beta_u_sqr + B_sqr * w_u_sqr
Q2 = A_sqr * Beta_v_sqr + B_sqr * w_u_sqr
Q3 = A_sqr * Beta_u_sqr + B_sqr * w_v_sqr
Q4 = A_sqr * Beta_v_sqr + B_sqr * w_v_sqr
Q5 = Beta_u_sqr + Beta_v_sqr

gamma1 = rho_sqr / Q1
gamma2 = rho_sqr / Q2
gamma3 = rho_sqr / Q3
gamma4 = rho_sqr / Q4
gamma5 = rho_sqr / (Q5 * C_sqr)

Work_1 = A_u * amp_u * amp_u * A_sqr * B_sqr * B_sqr * w_u_sqr * w_u_sqr / (Q1 * Q1)
Work_2 = A_v * amp_v * amp_v * A_sqr * B_sqr * B_sqr * w_v_sqr * w_v_sqr / (Q3 * Q3) * exp(-gamma1)
Work_3 = -Input_rec% qs * alpha_v * Beta_u_sqr * Beta_v_sqr * rho_sqr / (Q5 * Q5 * Q5) * exp(-gamma5)
Work_4 = A_v * amp_v * amp_v * A_sqr * A_sqr * B_sqr * Beta_u_sqr * w_v_sqr * rho_sqr / (Q3 * Q3 * Q3) * exp(-gamma3)
Work_5 = -Input_rec%qs * (alpha_u * I4 + alpha_v * C_sqr * Beta_v_sqr * Beta_v_sqr / (Q5 * Q5) * exp(-gamma5))
Work_6 = -2.0_10 * Input_rec%Nu * alpha_v * Beta_v_sqr / (Q5 * Q5 * Q5) * exp(-gamma5)
Work_7 = Beta_v_sqr - Beta_u_sqr - (Beta_v_sqr - 3.0_10 * Beta_u_sqr) * rho_sqr / (C_sqr * Q5)
Work_7 = Work_7 - Beta_u_sqr * rho_sqr * rho_sqr / (C_sqr * C_sqr * Q5 * Q5)
Work_8 = Work_6 * Work_7

Func_1 = Work_1 + Work_2 + Work_3 + Work_4 + Work_5 + Work_8

end Function Func_1

real (kind=10) Function Grad_F1_alpha_u(alpha_u, alpha_v, beta_u, beta_v, input_rec)
implicit none

Type Input_data
   Integer :: N_x, N_y, N_steps, NS_x, NS_y, Imid, IDebug, Ncont, Reg_w1
   Real (kind=10) :: T_step, T_initial, T_final
   Real (kind=10) :: Space_x, Space_y
   Real (kind=10) :: X_min, X_max, Y_min, Y_max
   Real (kind=10) :: Amp_1, Amp_2, Width_1, Width_2
   Real (kind=10) :: X_d1, Y_d1, Vel_1, Target_x
   Real (kind=10) :: X_d2, Y_d2, Vel_2
   Real (kind=10) :: Diff_1, Diff_2, AAAA, BBBB
   Real (kind=10) :: Nu, Theta, qs, R_phase, EPS, Om
   Real (kind=10) :: N_dp1, N_dp2, Damp_1, Damp_2, SRF
   Real (kind=10) :: Phi, Pi
   Real (kind=10) :: alpha1, alpha2, beta1, beta2, grad, inter, w1, w2, Reg_amt
End Type

Real (kind=10), intent(in) :: alpha_u, alpha_v, beta_u, beta_v
Type (Input_data), intent(in) :: input_rec
Real (kind=10) :: I4 = 0.2954314537_10, I42 = 0.1090862907_10

Grad_F1_alpha_u = -Input_rec%qs * I4
```

```
end Function Grad_F1_alpha_u

real (kind=10) Function Grad_F1_beta_u(alpha_u, alpha_v, beta_u, beta_v, input_rec)
implicit none

Type Input_data
   Integer :: N_x, N_y, N_steps, NS_x, NS_y, Imid, IDebug, Ncont, Reg_w1
   Real (kind=10) :: T_step, T_initial, T_final
   Real (kind=10) :: Space_x, Space_y
   Real (kind=10) :: X_min, X_max, Y_min, Y_max
   Real (kind=10) :: Amp_1, Amp_2, Width_1, Width_2
   Real (kind=10) :: X_d1, Y_d1, Vel_1, Target_x
   Real (kind=10) :: X_d2, Y_d2, Vel_2
   Real (kind=10) :: Diff_1, Diff_2, AAAA, BBBB
   Real (kind=10) :: Nu, Theta, qs, R_phase, EPS, Om
   Real (kind=10) :: N_dp1, N_dp2, Damp_1, Damp_2, SRF
   Real (kind=10) :: Phi, Pi
   Real (kind=10) :: alpha1, alpha2, beta1, beta2, grad, inter, w1, w2, Reg_amt
End Type

Real (kind=10), intent(in) :: alpha_u, alpha_v, beta_u, beta_v
Type (Input_data), intent(in) :: input_rec
Real (kind=10) :: A_u, A_v, amp_u, amp_v, w_u, w_v, w_u_sqr, w_v_sqr, amp_u_sqr, amp_v_sqr
Real (kind=10) :: Const_A = 0.8429538023_10, Const_B = 1.177410023_10, Const_C = 1.1_10
Real (kind=10) :: A_sqr, B_sqr, C_sqr, Beta_u_sqr, Beta_v_sqr
Real (kind=10) :: Work_1, Work_2, Work_3, Work_4, Work_5, Work_6, Work_7, Work_8, Work_9, work_10
Real (kind=10) :: Q1, Q2, Q3, Q4, Q5, rho, rho_sqr
Real (kind=10) :: gamma1, gamma2, gamma3, gamma4, gamma5

A_u = Input_rec%AAAA
A_v = Input_rec%BBBB
amp_u = Input_rec%Amp_1
amp_v = Input_rec%Amp_2
w_u = Input_rec%Width_1
w_v = Input_rec%Width_2
w_u_sqr = w_u * w_u
w_v_sqr = w_v * w_v
rho = Input_rec%X_d1 - Input_rec%X_d2
rho_sqr = rho * rho
A_sqr = Const_A * Const_A
B_sqr = Const_B * Const_B
C_sqr = Const_C * Const_C
Beta_u_sqr = beta_u * beta_u
Beta_v_sqr = beta_v * beta_v

Q1 = A_sqr * Beta_u_sqr + B_sqr * w_u_sqr
Q2 = A_sqr * Beta_v_sqr + B_sqr * w_u_sqr
Q3 = A_sqr * Beta_u_sqr + B_sqr * w_v_sqr
Q4 = A_sqr * Beta_v_sqr + B_sqr * w_v_sqr
Q5 = Beta_u_sqr + Beta_v_sqr

gamma1 = rho_sqr / Q1
gamma2 = rho_sqr / Q2
gamma3 = rho_sqr / Q3
gamma4 = rho_sqr / Q4
gamma5 = rho_sqr / (Q5 * C_sqr)

Work_1 = -4.0_10 * A_sqr * Beta_u * A_u * amp_u * amp_u * A_sqr * B_sqr * B_sqr * w_u_sqr * w_u_sqr / (Q1 * Q1 * Q1)
Work_2 = -2.0_10 * A_v * amp_v * amp_v * A_sqr * A_sqr * B_sqr * B_sqr * w_v_sqr * w_v_sqr / (Q3 * Q3) * exp(-gamma1)
Work_2 = Work_2 * (2.0_10 / Q3 - rho_sqr / (Q1 * Q1)) * Beta_u
Work_3 = -2.0_10 * Input_rec% qs * alpha_v * Beta_u * Beta_v_sqr * rho_sqr / (Q5 * Q5 * Q5) * exp(-gamma5)
Work_3 = Work_3 * (1.0_10 - 3.0_10 * Beta_u_sqr / Q5 + Beta_u_sqr * rho_sqr / (C_sqr * Q5 * Q5))
Work_4 =  2.0_10 * A_v * amp_v * amp_v * A_sqr * A_sqr * B_sqr * Beta_u * w_v_sqr * rho_sqr / (Q3 * Q3 * Q3)
Work_4 = Work_4 * exp(-gamma3) * (1.0_10 - Beta_u_sqr / Q3 * A_sqr * (3.0_10 - rho_sqr / Q3))
Work_5 = 2.0_10 * Input_rec%qs * alpha_v * C_sqr * Beta_v_sqr * Beta_v_sqr * Beta_u / (Q5 * Q5 * Q5) * exp(-gamma5)
Work_5 = Work_5 *( 2.0_10 - rho_sqr / (C_sqr * Q5))
Work_6 = -4.0_10 * Input_rec%Nu * alpha_v * Beta_v_sqr * Beta_u / (Q5 * Q5 * Q5) * exp(-gamma5)
Work_7 = (Beta_v_sqr - Beta_u_sqr - (Beta_v_sqr - 3.0_10 * Beta_u_sqr) * gamma5 - Beta_u_sqr * gamma5 * gamma5)
Work_10 = -3.0_10 / Q5 * Work_7
Work_8 = -1.0_10 + 3.0_10 * gamma5 + (Beta_v_sqr - 3.0_10 * Beta_u_sqr) * gamma5 / Q5 - gamma5 * gamma5
Work_8 = Work_8 + 2.0_10 * Beta_u_sqr * gamma5 * gamma5 / Q5
Work_9 = Work_7 * gamma5 / Q5
```

```
Grad_F1_beta_u = Work_1 + Work_2 + Work_3 + Work_4 + Work_5 + Work_6 * (Work_10 + Work_8 + Work_9)

end Function Grad_F1_beta_u

real (kind=10) Function Grad_F1_alpha_v(alpha_u, alpha_v, beta_u, beta_v, input_rec)
implicit none

Type Input_data
   Integer :: N_x, N_y, N_steps, NS_x, NS_y, Imid, IDebug, Ncont, Reg_w1
   Real (kind=10) :: T_step, T_initial, T_final
   Real (kind=10) :: Space_x, Space_y
   Real (kind=10) :: X_min, X_max, Y_min, Y_max
   Real (kind=10) :: Amp_1, Amp_2, Width_1, Width_2
   Real (kind=10) :: X_d1, Y_d1, Vel_1, Target_x
   Real (kind=10) :: X_d2, Y_d2, Vel_2
   Real (kind=10) :: Diff_1, Diff_2, AAAA, BBBB
   Real (kind=10) :: Nu, Theta, qs, R_phase, EPS, Om
   Real (kind=10) :: N_dp1, N_dp2, Damp_1, Damp_2, SRF
   Real (kind=10) :: Phi, Pi
   Real (kind=10) :: alpha1, alpha2, beta1, beta2, grad, inter, w1, w2, Reg_amt
End Type

Real (kind=10), intent(in) :: alpha_u, alpha_v, beta_u, beta_v
Type (Input_data), intent(in) :: input_rec
Real (kind=10) :: A_u, A_v, amp_u, amp_v, w_u, w_v, w_u_sqr, w_v_sqr, amp_u_sqr, amp_v_sqr
Real (kind=10) :: Const_A = 0.8429538023_10, Const_B = 1.177410023_10, Const_C = 1.1_10
Real (kind=10) :: A_sqr, B_sqr, C_sqr, Beta_u_sqr, Beta_v_sqr
Real (kind=10) :: Work_3, Work_5, Work_6, Work_7, Work_8
Real (kind=10) :: Q5, rho, rho_sqr
Real (kind=10) :: gamma5

A_u = Input_rec%AAAA
A_v = Input_rec%BBBB
amp_u = Input_rec%Amp_1
amp_v = Input_rec%Amp_2
w_u = Input_rec%Width_1
w_v = Input_rec%Width_2
w_u_sqr = w_u * w_u
w_v_sqr = w_v * w_v
rho = Input_rec%X_d1 - Input_rec%X_d2
rho_sqr = rho * rho
A_sqr = Const_A * Const_A
B_sqr = Const_B * Const_B
C_sqr = Const_C * Const_C
Beta_u_sqr = beta_u * beta_u
Beta_v_sqr = beta_v * beta_v

Q5 = Beta_u_sqr + Beta_v_sqr

gamma5 = rho_sqr / (Q5 * C_sqr)

Work_3 = -Input_rec% qs * Beta_u_sqr * Beta_v_sqr * rho_sqr / (Q5 * Q5 * Q5) * exp(-gamma5)
Work_5 = -Input_rec%qs * C_sqr * Beta_v_sqr * Beta_v_sqr / (Q5 * Q5) * exp(-gamma5)
Work_6 = -2.0_10 * Input_rec%Nu * Beta_v_sqr / (Q5 * Q5 * Q5) * exp(-gamma5)
Work_7 = Beta_v_sqr - Beta_u_sqr - (Beta_v_sqr - 3.0_10 * Beta_u_sqr) * rho_sqr / (C_sqr * Q5)
Work_7 = Work_7 - Beta_u_sqr * rho_sqr * rho_sqr / (C_sqr * C_sqr * Q5 * Q5)
Work_8 = Work_6 * Work_7

Grad_F1_alpha_v = Work_3 + Work_5 + Work_8

end Function Grad_F1_alpha_v

real (kind=10) Function Grad_F1_beta_v(alpha_u, alpha_v, beta_u, beta_v, input_rec)
implicit none

Type Input_data
   Integer :: N_x, N_y, N_steps, NS_x, NS_y, Imid, IDebug, Ncont, Reg_w1
   Real (kind=10) :: T_step, T_initial, T_final
   Real (kind=10) :: Space_x, Space_y
   Real (kind=10) :: X_min, X_max, Y_min, Y_max
   Real (kind=10) :: Amp_1, Amp_2, Width_1, Width_2
   Real (kind=10) :: X_d1, Y_d1, Vel_1, Target_x
   Real (kind=10) :: X_d2, Y_d2, Vel_2
   Real (kind=10) :: Diff_1, Diff_2, AAAA, BBBB
```

```
   Real (kind=10) :: Nu, Theta, qs, R_phase, EPS, Om
   Real (kind=10) :: N_dp1, N_dp2, Damp_1, Damp_2, SRF
   Real (kind=10) :: Phi, Pi
   Real (kind=10) :: alpha1, alpha2, beta1, beta2, grad, inter, w1, w2, Reg_amt
End Type

Real (kind=10), intent(in) :: alpha_u, alpha_v, beta_u, beta_v
Type (Input_data), intent(in) :: input_rec
Real (kind=10) :: A_u, A_v, amp_u, amp_v, w_u, w_v, w_u_sqr, w_v_sqr, amp_u_sqr, amp_v_sqr
Real (kind=10) :: Const_A = 0.8429538023_10, Const_B = 1.177410023_10, Const_C = 1.1_10
Real (kind=10) :: A_sqr, B_sqr, C_sqr, Beta_u_sqr, Beta_v_sqr
Real (kind=10) :: Work_1, Work_2, Work_3, Work_4, Work_5, Work_6, Work_7, Work_8, Work_9
Real (kind=10) :: Q1, Q2, Q3, Q4, Q5, rho, rho_sqr
Real (kind=10) :: gamma1, gamma2, gamma3, gamma4, gamma5

A_u = Input_rec%AAAA
A_v = Input_rec%BBBB
amp_u = Input_rec%Amp_1
amp_v = Input_rec%Amp_2
w_u = Input_rec%Width_1
w_v = Input_rec%Width_2
w_u_sqr = w_u * w_u
w_v_sqr = w_v * w_v
rho = Input_rec%X_d1 - Input_rec%X_d2
rho_sqr = rho * rho
A_sqr = Const_A * Const_A
B_sqr = Const_B * Const_B
C_sqr = Const_C * Const_C
Beta_u_sqr = beta_u * beta_u
Beta_v_sqr = beta_v * beta_v

Q1 = A_sqr * Beta_u_sqr + B_sqr * w_u_sqr
Q2 = A_sqr * Beta_v_sqr + B_sqr * w_u_sqr
Q3 = A_sqr * Beta_u_sqr + B_sqr * w_v_sqr
Q4 = A_sqr * Beta_v_sqr + B_sqr * w_v_sqr
Q5 = Beta_u_sqr + Beta_v_sqr

gamma1 = rho_sqr / Q1
gamma2 = rho_sqr / Q2
gamma3 = rho_sqr / Q3
gamma4 = rho_sqr / Q4
gamma5 = rho_sqr / (Q5 * C_sqr)

Work_1 = -2.0_10 * Input_rec% qs * alpha_v * Beta_u_sqr * Beta_v * rho_sqr / (Q5 * Q5 * Q5) * exp(-gamma5)
Work_1 = Work_1 * (1.0_10 - 3.0_10 * Beta_v_sqr / Q5 + Beta_v_sqr * gamma5 / Q5)

Work_2 = -2.0_10 * Input_rec%qs * alpha_v * C_sqr * Beta_v_sqr * Beta_v / (Q5 * Q5) * exp(-gamma5)
Work_2 = Work_2 *( 2.0_10 - Beta_v_sqr / Q5 * (2.0_10 - gamma5))

Work_3 = -4.0_10 * Input_rec%Nu * alpha_v * Beta_v / (Q5 * Q5 * Q5) * exp(-gamma5)
Work_4 = Beta_v_sqr - Beta_u_sqr - (Beta_v_sqr - 3.0_10 * Beta_u_sqr) * gamma5 - Beta_u_sqr * gamma5 * gamma5
Work_5 = 1.0_10 - beta_v_sqr / Q5 * (3.0_10 - gamma5)
Work_5 = Work_3 * Work_4 * Work_5

Work_6 = -2.0_10 * Input_rec%Nu * alpha_v * Beta_v_sqr / (Q5 * Q5 * Q5) * exp(-gamma5)
Work_7 = 2.0_10 * beta_v * (1.0_10 - gamma5)
Work_7 = Work_7 + 2.0_10 * beta_v * gamma5 / Q5 * (beta_v_sqr - 3.0_10 * beta_u_sqr + 2.0_10 * beta_u_sqr * gamma5)
Work_8 = Work_6 * Work_7

Grad_F1_beta_v = Work_1 + Work_2 + Work_5 + Work_8

end Function Grad_F1_beta_v

real (kind=10) Function Func_2(alpha_u, alpha_v, beta_u, beta_v, input_rec)
implicit none

Type Input_data
   Integer :: N_x, N_y, N_steps, NS_x, NS_y, Imid, IDebug, Ncont, Reg_w1
   Real (kind=10) :: T_step, T_initial, T_final
   Real (kind=10) :: Space_x, Space_y
   Real (kind=10) :: X_min, X_max, Y_min, Y_max
   Real (kind=10) :: Amp_1, Amp_2, Width_1, Width_2
   Real (kind=10) :: X_d1, Y_d1, Vel_1, Target_x
   Real (kind=10) :: X_d2, Y_d2, Vel_2
```

```
         Real (kind=10) :: Diff_1, Diff_2, AAAA, BBBB
         Real (kind=10) :: Nu, Theta, qs, R_phase, EPS, Om
         Real (kind=10) :: N_dp1, N_dp2, Damp_1, Damp_2, SRF
         Real (kind=10) :: Phi, Pi
         Real (kind=10) :: alpha1, alpha2, beta1, beta2, grad, inter, w1, w2, Reg_amt
      End Type

      Real (kind=10), intent(in) :: alpha_u, alpha_v, beta_u, beta_v
      Type (Input_data), intent(in) :: input_rec
      Real (kind=10) :: A_u, A_v, amp_u, amp_v, w_u, w_v, w_u_sqr, w_v_sqr, amp_u_sqr, amp_v_sqr
      Real (kind=10) :: Const_A = 0.8429538023_10, Const_B = 1.177410023_10, Const_C = 1.1_10
      Real (kind=10) :: I4 = 0.2954314537_10, I42 = 0.1090862907_10
      Real (kind=10) :: A_sqr, B_sqr, C_sqr, Beta_u_sqr, Beta_v_sqr
      Real (kind=10) :: Work_1, Work_2, Work_3, Work_4, Work_5, Work_6, Work_7, Work_8
      Real (kind=10) :: Q1, Q2, Q3, Q4, Q5, rho, rho_sqr
      Real (kind=10) :: gamma1, gamma2, gamma3, gamma4, gamma5

      A_u = Input_rec%AAAA
      A_v = Input_rec%BBBB
      amp_u = Input_rec%Amp_1
      amp_v = Input_rec%Amp_2
      amp_u_sqr = amp_u * amp_u
      amp_v_sqr = amp_v * amp_v
      w_u = Input_rec%Width_1
      w_v = Input_rec%Width_2
      w_u_sqr = w_u * w_u
      w_v_sqr = w_v * w_v
      rho = Input_rec%X_d1 - Input_rec%X_d2
      rho_sqr = rho * rho
      A_sqr = Const_A * Const_A
      B_sqr = Const_B * Const_B
      C_sqr = Const_C * Const_C
      Beta_u_sqr = beta_u * beta_u
      Beta_v_sqr = beta_v * beta_v

      Q1 = A_sqr * Beta_u_sqr + B_sqr * w_u_sqr
      Q2 = A_sqr * Beta_v_sqr + B_sqr * w_u_sqr
      Q3 = A_sqr * Beta_u_sqr + B_sqr * w_v_sqr
      Q4 = A_sqr * Beta_v_sqr + B_sqr * w_v_sqr
      Q5 = Beta_u_sqr + Beta_v_sqr

      gamma1 = rho_sqr / Q1
      gamma2 = rho_sqr / Q2
      gamma3 = rho_sqr / Q3
      gamma4 = rho_sqr / Q4
      gamma5 = rho_sqr / (Q5 * C_sqr)

      Work_1 = 2.0_10 * alpha_u * (2.0_10 * Input_rec%Nu * I42 + Input_rec%qs * Beta_u_sqr * I4)
!      Print*, 'Work_1 ', Work_1, 'Beta_v ', Beta_v
      Work_2 = - A_u * amp_u_sqr * A_sqr * B_sqr * w_u_sqr * Beta_u_sqr / Q1
!      Print*, 'Work_2 ', Work_2, 'Beta_v ', Beta_v
      Work_3 = A_v * amp_v_sqr * A_sqr * B_sqr * Beta_u_sqr * w_v_sqr / Q3 * exp(-gamma3)
!      Print*, 'Work_3 ', Work_3, 'Beta_v ', Beta_v
      Work_4 = -2.0_10 * Input_rec%Nu * alpha_v * Beta_u_sqr * Beta_v_sqr / (Q5 * Q5) * (1.0_10 - gamma5) * exp(-gamma5)
!      Print*, 'Work_4 ', Work_4, 'Beta_v ', Beta_v
      Work_5 = Input_rec%qs * alpha_v * C_sqr * Beta_u_sqr * Beta_v_sqr / Q5 * exp(-gamma5)
!      Print*, 'Work_5 ', Work_5, 'Beta_v ', Beta_v

      Func_2 = Work_1 + Work_2 + Work_3 + Work_4 + Work_5

      end Function Func_2

      real (kind=10) Function Grad_F2_alpha_u(alpha_u, alpha_v, beta_u, beta_v, input_rec)
      implicit none

      Type Input_data
         Integer :: N_x, N_y, N_steps, NS_x, NS_y, Imid, IDebug, Ncont, Reg_w1
         Real (kind=10) :: T_step, T_initial, T_final
         Real (kind=10) :: Space_x, Space_y
         Real (kind=10) :: X_min, X_max, Y_min, Y_max
         Real (kind=10) :: Amp_1, Amp_2, Width_1, Width_2
         Real (kind=10) :: X_d1, Y_d1, Vel_1, Target_x
         Real (kind=10) :: X_d2, Y_d2, Vel_2
         Real (kind=10) :: Diff_1, Diff_2, AAAA, BBBB
```

```
   Real (kind=10) :: Nu, Theta, qs, R_phase, EPS, Om
   Real (kind=10) :: N_dp1, N_dp2, Damp_1, Damp_2, SRF
   Real (kind=10) :: Phi, Pi
   Real (kind=10) :: alpha1, alpha2, beta1, beta2, grad, inter, w1, w2, Reg_amt
End Type

Real (kind=10), intent(in) :: alpha_u, alpha_v, beta_u, beta_v
Type (Input_data), intent(in) :: input_rec
Real (kind=10) :: A_u, A_v, amp_u, amp_v, w_u, w_v, w_u_sqr, w_v_sqr, amp_u_sqr, amp_v_sqr
Real (kind=10) :: Const_A = 0.8429538023_10, Const_B = 1.177410023_10, Const_C = 1.1_10
Real (kind=10) :: I4 = 0.2954314537_10, I42 = 0.1090862907_10
Real (kind=10) :: A_sqr, B_sqr, C_sqr, Beta_u_sqr, Beta_v_sqr
Real (kind=10) :: Work_1, Work_2, Work_3, Work_4, Work_5, Work_6, Work_7, Work_8
Real (kind=10) :: Q1, Q2, Q3, Q4, Q5, rho, rho_sqr
Real (kind=10) :: gamma1, gamma2, gamma3, gamma4, gamma5

A_u = Input_rec%AAAA
A_v = Input_rec%BBBB
amp_u = Input_rec%Amp_1
amp_v = Input_rec%Amp_2
amp_u_sqr = amp_u * amp_u
amp_v_sqr = amp_v * amp_v
w_u = Input_rec%Width_1
w_v = Input_rec%Width_2
w_u_sqr = w_u * w_u
w_v_sqr = w_v * w_v
rho = Input_rec%X_d1 - Input_rec%X_d2
rho_sqr = rho * rho
A_sqr = Const_A * Const_A
B_sqr = Const_B * Const_B
C_sqr = Const_C * Const_C
Beta_u_sqr = beta_u * beta_u
Beta_v_sqr = beta_v * beta_v

Q1 = A_sqr * Beta_u_sqr + B_sqr * w_u_sqr
Q2 = A_sqr * Beta_v_sqr + B_sqr * w_u_sqr
Q3 = A_sqr * Beta_u_sqr + B_sqr * w_v_sqr
Q4 = A_sqr * Beta_v_sqr + B_sqr * w_v_sqr
Q5 = Beta_u_sqr + Beta_v_sqr

gamma1 = rho_sqr / Q1
gamma2 = rho_sqr / Q2
gamma3 = rho_sqr / Q3
gamma4 = rho_sqr / Q4
gamma5 = rho_sqr / (Q5 * C_sqr)

Grad_F2_alpha_u = 4.0_10 * Input_rec%Nu * I42 + 2.0_10 * Input_rec%qs * Beta_u_sqr * I4

end Function Grad_F2_alpha_u

real (kind=10) Function Grad_F2_alpha_v(alpha_u, alpha_v, beta_u, beta_v, input_rec)
implicit none

Type Input_data
   Integer :: N_x, N_y, N_steps, NS_x, NS_y, Imid, IDebug, Ncont, Reg_w1
   Real (kind=10) :: T_step, T_initial, T_final
   Real (kind=10) :: Space_x, Space_y
   Real (kind=10) :: X_min, X_max, Y_min, Y_max
   Real (kind=10) :: Amp_1, Amp_2, Width_1, Width_2
   Real (kind=10) :: X_d1, Y_d1, Vel_1, Target_x
   Real (kind=10) :: X_d2, Y_d2, Vel_2
   Real (kind=10) :: Diff_1, Diff_2, AAAA, BBBB
   Real (kind=10) :: Nu, Theta, qs, R_phase, EPS, Om
   Real (kind=10) :: N_dp1, N_dp2, Damp_1, Damp_2, SRF
   Real (kind=10) :: Phi, Pi
   Real (kind=10) :: alpha1, alpha2, beta1, beta2, grad, inter, w1, w2, Reg_amt
End Type

Real (kind=10), intent(in) :: alpha_u, alpha_v, beta_u, beta_v
Type (Input_data), intent(in) :: input_rec
Real (kind=10) :: A_u, A_v, amp_u, amp_v, w_u, w_v, w_u_sqr, w_v_sqr, amp_u_sqr, amp_v_sqr
Real (kind=10) :: Const_A = 0.8429538023_10, Const_B = 1.177410023_10, Const_C = 1.1_10
Real (kind=10) :: I4 = 0.2954314537_10, I42 = 0.1090862907_10
Real (kind=10) :: A_sqr, B_sqr, C_sqr, Beta_u_sqr, Beta_v_sqr
```

```
Real (kind=10) :: Work_1, Work_2, Work_3, Work_4, Work_5, Work_6, Work_7, Work_8
Real (kind=10) :: Q1, Q2, Q3, Q4, Q5, rho, rho_sqr
Real (kind=10) :: gamma1, gamma2, gamma3, gamma4, gamma5

A_u = Input_rec%AAAA
A_v = Input_rec%BBBB
amp_u = Input_rec%Amp_1
amp_v = Input_rec%Amp_2
amp_u_sqr = amp_u * amp_u
amp_v_sqr = amp_v * amp_v
w_u = Input_rec%Width_1
w_v = Input_rec%Width_2
w_u_sqr = w_u * w_u
w_v_sqr = w_v * w_v
rho = Input_rec%X_d1 - Input_rec%X_d2
rho_sqr = rho * rho
A_sqr = Const_A * Const_A
B_sqr = Const_B * Const_B
C_sqr = Const_C * Const_C
Beta_u_sqr = beta_u * beta_u
Beta_v_sqr = beta_v * beta_v

Q1 = A_sqr * Beta_u_sqr + B_sqr * w_u_sqr
Q2 = A_sqr * Beta_v_sqr + B_sqr * w_u_sqr
Q3 = A_sqr * Beta_u_sqr + B_sqr * w_v_sqr
Q4 = A_sqr * Beta_v_sqr + B_sqr * w_v_sqr
Q5 = Beta_u_sqr + Beta_v_sqr

gamma1 = rho_sqr / Q1
gamma2 = rho_sqr / Q2
gamma3 = rho_sqr / Q3
gamma4 = rho_sqr / Q4
gamma5 = rho_sqr / (Q5 * C_sqr)

Work_4 = -2.0_10 * Input_rec%Nu * Beta_u_sqr * Beta_v_sqr / (Q5 * Q5) * (1.0_10 - gamma5) * exp(-gamma5)
Work_5 = Input_rec%qs * C_sqr * Beta_u_sqr * Beta_v_sqr / Q5 * exp(-gamma5)

Grad_F2_alpha_v = Work_4 + Work_5

end Function Grad_F2_alpha_v

real (kind=10) Function Grad_F2_beta_u(alpha_u, alpha_v, beta_u, beta_v, input_rec)
implicit none

Type Input_data
   Integer :: N_x, N_y, N_steps, NS_x, NS_y, Imid, IDebug, Ncont, Reg_w1
   Real (kind=10) :: T_step, T_initial, T_final
   Real (kind=10) :: Space_x, Space_y
   Real (kind=10) :: X_min, X_max, Y_min, Y_max
   Real (kind=10) :: Amp_1, Amp_2, Width_1, Width_2
   Real (kind=10) :: X_d1, Y_d1, Vel_1, Target_x
   Real (kind=10) :: X_d2, Y_d2, Vel_2
   Real (kind=10) :: Diff_1, Diff_2, AAAA, BBBB
   Real (kind=10) :: Nu, Theta, qs, R_phase, EPS, Om
   Real (kind=10) :: N_dp1, N_dp2, Damp_1, Damp_2, SRF
   Real (kind=10) :: Phi, Pi
   Real (kind=10) :: alpha1, alpha2, beta1, beta2, grad, inter, w1, w2, Reg_amt
End Type

Real (kind=10), intent(in) :: alpha_u, alpha_v, beta_u, beta_v
Type (Input_data), intent(in) :: input_rec
Real (kind=10) :: A_u, A_v, amp_u, amp_v, w_u, w_v, w_u_sqr, w_v_sqr, amp_u_sqr, amp_v_sqr
Real (kind=10) :: Const_A = 0.8429538023_10, Const_B = 1.177410023_10, Const_C = 1.1_10
Real (kind=10) :: I4 = 0.2954314537_10, I42 = 0.1090862907_10
Real (kind=10) :: A_sqr, B_sqr, C_sqr, Beta_u_sqr, Beta_v_sqr
Real (kind=10) :: Work_1, Work_2, Work_3, Work_4, Work_5, Work_6, Work_7, Work_8
Real (kind=10) :: Q1, Q2, Q3, Q4, Q5, rho, rho_sqr
Real (kind=10) :: gamma1, gamma2, gamma3, gamma4, gamma5

A_u = Input_rec%AAAA
A_v = Input_rec%BBBB
amp_u = Input_rec%Amp_1
amp_v = Input_rec%Amp_2
amp_u_sqr = amp_u * amp_u
```

```
amp_v_sqr = amp_v * amp_v
w_u = Input_rec%Width_1
w_v = Input_rec%Width_2
w_u_sqr = w_u * w_u
w_v_sqr = w_v * w_v
rho = Input_rec%X_d1 - Input_rec%X_d2
rho_sqr = rho * rho
A_sqr = Const_A * Const_A
B_sqr = Const_B * Const_B
C_sqr = Const_C * Const_C
Beta_u_sqr = beta_u * beta_u
Beta_v_sqr = beta_v * beta_v

Q1 = A_sqr * Beta_u_sqr + B_sqr * w_u_sqr
Q2 = A_sqr * Beta_v_sqr + B_sqr * w_u_sqr
Q3 = A_sqr * Beta_u_sqr + B_sqr * w_v_sqr
Q4 = A_sqr * Beta_v_sqr + B_sqr * w_v_sqr
Q5 = Beta_u_sqr + Beta_v_sqr

gamma1 = rho_sqr / Q1
gamma2 = rho_sqr / Q2
gamma3 = rho_sqr / Q3
gamma4 = rho_sqr / Q4
gamma5 = rho_sqr / (Q5 * C_sqr)

Work_1 = -2.0_10 * A_u * amp_u_sqr * A_sqr * B_sqr * w_u_sqr * Beta_u / Q1 * (1.0_10 - A_sqr * Beta_u_sqr / Q1)
Work_1 = Work_1 + 4.0_10 * alpha_u * Input_rec%qs * Beta_u * I4

Work_2 = 2.0_10 * A_v * amp_v_sqr * A_sqr * B_sqr * w_v_sqr * Beta_u / Q3 * exp(-gamma3)
Work_3 = 1.0_10 - Beta_u_sqr * A_sqr / Q3 + A_sqr * Beta_u_sqr * rho_sqr / (Q3 * Q3)
Work_2 = Work_2 * Work_3

Work_4 = -4.0_10 * Input_rec%Nu * alpha_v * Beta_u * Beta_v_sqr / (Q5 * Q5) * exp(-gamma5)
Work_5 = 1.0_10 - gamma5 - 2.0_10 * Beta_u_sqr * (1.0_10 - gamma5) / Q5 + Beta_u_sqr * gamma5 / Q5
Work_5 = Work_5 + Beta_u_sqr * (1.0_10 - gamma5) * gamma5 / Q5
Work_4 = Work_4 * Work_5

Work_6 = 2.0_10 * Input_rec%qs * alpha_v * Beta_v_sqr * Beta_u * exp(-gamma5) * C_sqr / Q5
Work_7 = 1.0_10 - Beta_u_sqr / Q5 + Beta_u_sqr * gamma5 / Q5
Work_6 = Work_6 * Work_7

Grad_F2_beta_u = Work_1 + Work_2 + Work_4 + Work_6

end Function Grad_F2_beta_u

real (kind=10) Function Grad_F2_beta_v(alpha_u, alpha_v, beta_u, beta_v, input_rec)
implicit none

Type Input_data
   Integer :: N_x, N_y, N_steps, NS_x, NS_y, Imid, IDebug, Ncont, Reg_w1
   Real (kind=10) :: T_step, T_initial, T_final
   Real (kind=10) :: Space_x, Space_y
   Real (kind=10) :: X_min, X_max, Y_min, Y_max
   Real (kind=10) :: Amp_1, Amp_2, Width_1, Width_2
   Real (kind=10) :: X_d1, Y_d1, Vel_1, Target_x
   Real (kind=10) :: X_d2, Y_d2, Vel_2
   Real (kind=10) :: Diff_1, Diff_2, AAAA, BBBB
   Real (kind=10) :: Nu, Theta, qs, R_phase, EPS, Om
   Real (kind=10) :: N_dp1, N_dp2, Damp_1, Damp_2, SRF
   Real (kind=10) :: Phi, Pi
   Real (kind=10) :: alpha1, alpha2, beta1, beta2, grad, inter, w1, w2, Reg_amt
End Type

Real (kind=10), intent(in) :: alpha_u, alpha_v, beta_u, beta_v
Type (Input_data), intent(in) :: input_rec
Real (kind=10) :: A_u, A_v, amp_u, amp_v, w_u, w_v, w_u_sqr, w_v_sqr, amp_u_sqr, amp_v_sqr
Real (kind=10) :: Const_A = 0.8429538023_10, Const_B = 1.177410023_10, Const_C = 1.1_10
Real (kind=10) :: I4 = 0.2954314537_10, I42 = 0.1090862907_10
Real (kind=10) :: A_sqr, B_sqr, C_sqr, Beta_u_sqr, Beta_v_sqr
Real (kind=10) :: Work_1, Work_2, Work_3, Work_4, Work_5, Work_6, Work_7, Work_8
Real (kind=10) :: Q1, Q2, Q3, Q4, Q5, rho, rho_sqr
Real (kind=10) :: gamma1, gamma2, gamma3, gamma4, gamma5

A_u = Input_rec%AAAA
```

```
      A_v = Input_rec%BBBB
      amp_u = Input_rec%Amp_1
      amp_v = Input_rec%Amp_2
      amp_u_sqr = amp_u * amp_u
      amp_v_sqr = amp_v * amp_v
      w_u = Input_rec%Width_1
      w_v = Input_rec%Width_2
      w_u_sqr = w_u * w_u
      w_v_sqr = w_v * w_v
      rho = Input_rec%X_d1 - Input_rec%X_d2
      rho_sqr = rho * rho
      A_sqr = Const_A * Const_A
      B_sqr = Const_B * Const_B
      C_sqr = Const_C * Const_C
      Beta_u_sqr = beta_u * beta_u
      Beta_v_sqr = beta_v * beta_v

      Q1 = A_sqr * Beta_u_sqr + B_sqr * w_u_sqr
      Q2 = A_sqr * Beta_v_sqr + B_sqr * w_u_sqr
      Q3 = A_sqr * Beta_u_sqr + B_sqr * w_v_sqr
      Q4 = A_sqr * Beta_v_sqr + B_sqr * w_v_sqr
      Q5 = Beta_u_sqr + Beta_v_sqr

      gamma1 = rho_sqr / Q1
      gamma2 = rho_sqr / Q2
      gamma3 = rho_sqr / Q3
      gamma4 = rho_sqr / Q4
      gamma5 = rho_sqr / Q5 / C_sqr

!      Work_1 = 2.0_10 * A_v * amp_v_sqr * A_sqr * B_sqr * w_v_sqr * Beta_v * exp(-gamma3) / Q3

      Work_3 = -4.0_10 * Input_rec%Nu * alpha_v * Beta_v * Beta_u_sqr / (Q5 * Q5) * exp(-gamma5)
!      Print*, 'Work_3 ', Work_3
      Work_4 = 1.0_10 - gamma5 - Beta_v_sqr * (2.0_10 - 4.0_10 * gamma5 + gamma5 * gamma5) / Q5
!      Print*, 'Work_4 ', Work_4
      Work_3 = Work_3 * Work_4

      Work_5 = 2.0_10 * Input_rec%qs * alpha_v * C_sqr * Beta_u_sqr * Beta_v * exp(-gamma5) / Q5
!      Print*, 'Work_5 ', Work_5
      Work_6 = 1.0_10 - Beta_v_sqr / Q5 * (1.0_10 - gamma5)
!      Print*, 'Work_6 ', Work_6
      Work_5 = Work_5 * Work_6

!      Print*, 'Work 3 ', Work_3, 'Work_5 ', Work_5, ' gamma5 ', gamma5, ' Q5 ', Q5

      Grad_F2_beta_v = Work_3 + work_5

      end Function Grad_F2_beta_v

      real (kind=10) Function Func_3(alpha_u, alpha_v, beta_u, beta_v, input_rec)
      implicit none

      Type Input_data
         Integer :: N_x, N_y, N_steps, NS_x, NS_y, Imid, IDebug, Ncont, Reg_w1
         Real (kind=10) :: T_step, T_initial, T_final
         Real (kind=10) :: Space_x, Space_y
         Real (kind=10) :: X_min, X_max, Y_min, Y_max
         Real (kind=10) :: Amp_1, Amp_2, Width_1, Width_2
         Real (kind=10) :: X_d1, Y_d1, Vel_1, Target_x
         Real (kind=10) :: X_d2, Y_d2, Vel_2
         Real (kind=10) :: Diff_1, Diff_2, AAAA, BBBB
         Real (kind=10) :: Nu, Theta, qs, R_phase, EPS, Om
         Real (kind=10) :: N_dp1, N_dp2, Damp_1, Damp_2, SRF
         Real (kind=10) :: Phi, Pi
         Real (kind=10) :: alpha1, alpha2, beta1, beta2, grad, inter, w1, w2, Reg_amt
      End Type

      Real (kind=10), intent(in) :: alpha_u, alpha_v, beta_u, beta_v
      Type (Input_data), intent(in) :: input_rec
      Real (kind=10) :: A_u, A_v, amp_u, amp_v, w_u, w_v, w_u_sqr, w_v_sqr, amp_u_sqr, amp_v_sqr
      Real (kind=10) :: Const_A = 0.8429538023_10, Const_B = 1.177410023_10, Const_C = 1.1_10
      Real (kind=10) :: I4 = 0.2954314537_10, I42 = 0.1090862907_10
      Real (kind=10) :: A_sqr, B_sqr, C_sqr, Beta_u_sqr, Beta_v_sqr
      Real (kind=10) :: Work_1, Work_2, Work_3, Work_4, Work_5, Work_6, Work_7, Work_8
```

```
Real (kind=10) :: Q1, Q2, Q3, Q4, Q5, rho, rho_sqr
Real (kind=10) :: gamma1, gamma2, gamma3, gamma4, gamma5

A_u = Input_rec%AAAA
A_v = Input_rec%BBBB
amp_u = Input_rec%Amp_1
amp_v = Input_rec%Amp_2
w_u = Input_rec%Width_1
w_v = Input_rec%Width_2
w_u_sqr = w_u * w_u
w_v_sqr = w_v * w_v
rho = Input_rec%X_d1 - Input_rec%X_d2
rho_sqr = rho * rho
A_sqr = Const_A * Const_A
B_sqr = Const_B * Const_B
C_sqr = Const_C * Const_C
Beta_u_sqr = beta_u * beta_u
Beta_v_sqr = beta_v * beta_v

Q1 = A_sqr * Beta_u_sqr + B_sqr * w_u_sqr
Q2 = A_sqr * Beta_v_sqr + B_sqr * w_u_sqr
Q3 = A_sqr * Beta_u_sqr + B_sqr * w_v_sqr
Q4 = A_sqr * Beta_v_sqr + B_sqr * w_v_sqr
Q5 = Beta_u_sqr + Beta_v_sqr

gamma1 = rho_sqr / Q1
gamma2 = rho_sqr / Q2
gamma3 = rho_sqr / Q3
gamma4 = rho_sqr / Q4
gamma5 = rho_sqr / (Q5 * C_sqr)

Work_1 = A_v * amp_v * amp_v * A_sqr * B_sqr * B_sqr * w_v_sqr * w_v_sqr / (Q4 * Q4)
Work_2 = A_u * amp_u * amp_u * A_sqr * B_sqr * B_sqr * w_u_sqr * w_u_sqr / (Q2 * Q2) * exp(-gamma4)
Work_3 = -Input_rec% qs * alpha_u * Beta_v_sqr * Beta_u_sqr * rho_sqr / (Q5 * Q5 * Q5) * exp(-gamma5)
Work_4 = A_u * amp_u * amp_u * A_sqr * A_sqr * B_sqr * Beta_v_sqr * w_u_sqr * rho_sqr / (Q2 * Q2 * Q2) * exp(-gamma2)
Work_5 = -Input_rec%qs * (alpha_v * I4 + alpha_u * C_sqr * Beta_u_sqr * Beta_u_sqr / (Q5 * Q5) * exp(-gamma5))
Work_6 = -2.0_10 * Input_rec%Nu * alpha_u * Beta_u_sqr / (Q5 * Q5 * Q5) * exp(-gamma5)
Work_7 = Beta_u_sqr - Beta_v_sqr - (Beta_u_sqr - 3.0_10 * Beta_v_sqr) * rho_sqr / (C_sqr * Q5)
Work_7 = Work_7 - Beta_v_sqr * rho_sqr * rho_sqr / (C_sqr * C_sqr * Q5 * Q5)
Work_8 = Work_6 * Work_7

Func_3 = Work_1 + Work_2 + Work_3 + Work_4 + Work_5 + Work_8

end Function Func_3

real (kind=10) Function Grad_F3_alpha_u(alpha_u, alpha_v, beta_u, beta_v, input_rec)
implicit none

Type Input_data
   Integer :: N_x, N_y, N_steps, NS_x, NS_y, Imid, IDebug, Ncont, Reg_w1
   Real (kind=10) :: T_step, T_initial, T_final
   Real (kind=10) :: Space_x, Space_y
   Real (kind=10) :: X_min, X_max, Y_min, Y_max
   Real (kind=10) :: Amp_1, Amp_2, Width_1, Width_2
   Real (kind=10) :: X_d1, Y_d1, Vel_1, Target_x
   Real (kind=10) :: X_d2, Y_d2, Vel_2
   Real (kind=10) :: Diff_1, Diff_2, AAAA, BBBB
   Real (kind=10) :: Nu, Theta, qs, R_phase, EPS, Om
   Real (kind=10) :: N_dp1, N_dp2, Damp_1, Damp_2, SRF
   Real (kind=10) :: Phi, Pi
   Real (kind=10) :: alpha1, alpha2, beta1, beta2, grad, inter, w1, w2, Reg_amt
End Type

Real (kind=10), intent(in) :: alpha_u, alpha_v, beta_u, beta_v
Type (Input_data), intent(in) :: input_rec
Real (kind=10) :: A_u, A_v, amp_u, amp_v, w_u, w_v, w_u_sqr, w_v_sqr, amp_u_sqr, amp_v_sqr
Real (kind=10) :: Const_A = 0.8429538023_10, Const_B = 1.177410023_10, Const_C = 1.1_10
Real (kind=10) :: A_sqr, B_sqr, C_sqr, Beta_u_sqr, Beta_v_sqr
Real (kind=10) :: Work_1, Work_2, Work_3, Work_4, Work_5, Work_6, Work_7, Work_8, Work_9, Work_10
Real (kind=10) :: Q1, Q2, Q3, Q4, Q5, rho, rho_sqr
Real (kind=10) :: gamma1, gamma2, gamma3, gamma4, gamma5

A_u = Input_rec%AAAA
A_v = Input_rec%BBBB
```

```
amp_u = Input_rec%Amp_1
amp_v = Input_rec%Amp_2
w_u = Input_rec%Width_1
w_v = Input_rec%Width_2
w_u_sqr = w_u * w_u
w_v_sqr = w_v * w_v
rho = Input_rec%X_d1 - Input_rec%X_d2
rho_sqr = rho * rho
A_sqr = Const_A * Const_A
B_sqr = Const_B * Const_B
C_sqr = Const_C * Const_C
Beta_u_sqr = beta_u * beta_u
Beta_v_sqr = beta_v * beta_v

Q1 = A_sqr * Beta_u_sqr + B_sqr * w_u_sqr
Q2 = A_sqr * Beta_v_sqr + B_sqr * w_u_sqr
Q3 = A_sqr * Beta_u_sqr + B_sqr * w_v_sqr
Q4 = A_sqr * Beta_v_sqr + B_sqr * w_v_sqr
Q5 = Beta_u_sqr + Beta_v_sqr

gamma1 = rho_sqr / Q1
gamma2 = rho_sqr / Q2
gamma3 = rho_sqr / Q3
gamma4 = rho_sqr / Q4
gamma5 = rho_sqr / (Q5 * C_sqr)

Work_3 = -Input_rec% qs * Beta_v_sqr * Beta_u_sqr * rho_sqr / (Q5 * Q5 * Q5) * exp(-gamma5)
Work_5 = -Input_rec%qs * C_sqr * Beta_u_sqr * Beta_u_sqr / (Q5 * Q5) * exp(-gamma5)
Work_6 = -2.0_10 * Input_rec%Nu * Beta_u_sqr / (Q5 * Q5 * Q5) * exp(-gamma5)
Work_7 = Beta_u_sqr - Beta_v_sqr - (Beta_u_sqr - 3.0_10 * Beta_v_sqr) * rho_sqr / (C_sqr * Q5)
Work_7 = Work_7 - Beta_v_sqr * rho_sqr * rho_sqr / (C_sqr * C_sqr * Q5 * Q5)
Work_8 = Work_6 * Work_7

Grad_F3_alpha_u = Work_3 + Work_5 + Work_8

end Function Grad_F3_alpha_u

real (kind=10) Function Grad_F3_beta_u(alpha_u, alpha_v, beta_u, beta_v, input_rec)
implicit none

Type Input_data
   Integer :: N_x, N_y, N_steps, NS_x, NS_y, Imid, IDebug, Ncont, Reg_w1
   Real (kind=10) :: T_step, T_initial, T_final
   Real (kind=10) :: Space_x, Space_y
   Real (kind=10) :: X_min, X_max, Y_min, Y_max
   Real (kind=10) :: Amp_1, Amp_2, Width_1, Width_2
   Real (kind=10) :: X_d1, Y_d1, Vel_1, Target_x
   Real (kind=10) :: X_d2, Y_d2, Vel_2
   Real (kind=10) :: Diff_1, Diff_2, AAAA, BBBB
   Real (kind=10) :: Nu, Theta, qs, R_phase, EPS, Om
   Real (kind=10) :: N_dp1, N_dp2, Damp_1, Damp_2, SRF
   Real (kind=10) :: Phi, Pi
   Real (kind=10) :: alpha1, alpha2, beta1, beta2, grad, inter, w1, w2, Reg_amt
End Type

Real (kind=10), intent(in) :: alpha_u, alpha_v, beta_u, beta_v
Type (Input_data), intent(in) :: input_rec
Real (kind=10) :: A_u, A_v, amp_u, amp_v, w_u, w_v, w_u_sqr, w_v_sqr, amp_u_sqr, amp_v_sqr
Real (kind=10) :: Const_A = 0.8429538023_10, Const_B = 1.177410023_10, Const_C = 1.1_10
Real (kind=10) :: A_sqr, B_sqr, C_sqr, Beta_u_sqr, Beta_v_sqr
Real (kind=10) :: Work_1, Work_2, Work_3, Work_4, Work_5, Work_6, Work_7, Work_8, Work_9, Work_10
Real (kind=10) :: Q1, Q2, Q3, Q4, Q5, rho, rho_sqr
Real (kind=10) :: gamma1, gamma2, gamma3, gamma4, gamma5

A_u = Input_rec%AAAA
A_v = Input_rec%BBBB
amp_u = Input_rec%Amp_1
amp_v = Input_rec%Amp_2
w_u = Input_rec%Width_1
w_v = Input_rec%Width_2
w_u_sqr = w_u * w_u
w_v_sqr = w_v * w_v
rho = Input_rec%X_d1 - Input_rec%X_d2
rho_sqr = rho * rho
```

```
A_sqr = Const_A * Const_A
B_sqr = Const_B * Const_B
C_sqr = Const_C * Const_C
Beta_u_sqr = beta_u * beta_u
Beta_v_sqr = beta_v * beta_v

Q1 = A_sqr * Beta_u_sqr + B_sqr * w_u_sqr
Q2 = A_sqr * Beta_v_sqr + B_sqr * w_u_sqr
Q3 = A_sqr * Beta_u_sqr + B_sqr * w_v_sqr
Q4 = A_sqr * Beta_v_sqr + B_sqr * w_v_sqr
Q5 = Beta_u_sqr + Beta_v_sqr

gamma1 = rho_sqr / Q1
gamma2 = rho_sqr / Q2
gamma3 = rho_sqr / Q3
gamma4 = rho_sqr / Q4
gamma5 = rho_sqr / (Q5 * C_sqr)

Work_1 = -2.0_10 * Input_rec% qs * alpha_u * Beta_v_sqr * Beta_u * rho_sqr / (Q5 * Q5 * Q5) * exp(-gamma5)
Work_1 = Work_1 * (1.0_10 - 3.0_10 * Beta_u_sqr / Q5 + Beta_u_sqr * gamma5 / Q5)

Work_2 = -2.0_10 * Input_rec%qs * alpha_u * C_sqr * Beta_u_sqr * Beta_u / (Q5 * Q5) * exp(-gamma5)
Work_2 = Work_2 *( 2.0_10 - Beta_u_sqr / Q5 * (2.0_10 - gamma5))

Work_3 = -4.0_10 * Input_rec%Nu * alpha_u * Beta_u / (Q5 * Q5 * Q5) * exp(-gamma5)
Work_4 = Beta_u_sqr - Beta_v_sqr - (Beta_u_sqr - 3.0_10 * Beta_v_sqr) * gamma5 - Beta_v_sqr * gamma5 * gamma5
Work_5 = 1.0_10 - beta_u_sqr / Q5 * (3.0_10 - gamma5)
Work_5 = Work_3 * Work_4 * Work_5

Work_6 = -2.0_10 * Input_rec%Nu * alpha_u * Beta_u_sqr / (Q5 * Q5 * Q5) * exp(-gamma5)
Work_7 = 2.0_10 * beta_u * (1.0_10 - gamma5)
Work_7 = Work_7 + 2.0_10 * beta_u * gamma5 / Q5 * (beta_u_sqr - 3.0_10 * beta_v_sqr + 2.0_10 * beta_v_sqr * gamma5)
Work_8 = Work_6 * Work_7

Grad_F3_beta_u = Work_1 + Work_2 + Work_5 + Work_8

end Function Grad_F3_beta_u

real (kind=10) Function Grad_F3_alpha_v(alpha_u, alpha_v, beta_u, beta_v, input_rec)
implicit none

Type Input_data
   Integer :: N_x, N_y, N_steps, NS_x, NS_y, Imid, IDebug, Ncont, Reg_w1
   Real (kind=10) :: T_step, T_initial, T_final
   Real (kind=10) :: Space_x, Space_y
   Real (kind=10) :: X_min, X_max, Y_min, Y_max
   Real (kind=10) :: Amp_1, Amp_2, Width_1, Width_2
   Real (kind=10) :: X_d1, Y_d1, Vel_1, Target_x
   Real (kind=10) :: X_d2, Y_d2, Vel_2
   Real (kind=10) :: Diff_1, Diff_2, AAAA, BBBB
   Real (kind=10) :: Nu, Theta, qs, R_phase, EPS, Om
   Real (kind=10) :: N_dp1, N_dp2, Damp_1, Damp_2, SRF
   Real (kind=10) :: Phi, Pi
   Real (kind=10) :: alpha1, alpha2, beta1, beta2, grad, inter, w1, w2, Reg_amt
End Type

Real (kind=10), intent(in) :: alpha_u, alpha_v, beta_u, beta_v
Type (Input_data), intent(in) :: input_rec
Real (kind=10) :: A_u, A_v, amp_u, amp_v, w_u, w_v, w_u_sqr, w_v_sqr, amp_u_sqr, amp_v_sqr
Real (kind=10) :: Const_A = 0.8429538023_10, Const_B = 1.177410023_10, Const_C = 1.1_10
Real (kind=10) :: I4 = 0.2954314537_10, I42 = 0.1090862907_10
Real (kind=10) :: A_sqr, B_sqr, C_sqr, Beta_u_sqr, Beta_v_sqr
Real (kind=10) :: Work_3, Work_5, Work_6, Work_7, Work_8
Real (kind=10) :: Q5, rho, rho_sqr
Real (kind=10) :: gamma5

A_u = Input_rec%AAAA
A_v = Input_rec%BBBB
amp_u = Input_rec%Amp_1
amp_v = Input_rec%Amp_2
w_u = Input_rec%Width_1
w_v = Input_rec%Width_2
w_u_sqr = w_u * w_u
w_v_sqr = w_v * w_v
```

```
rho = Input_rec%X_d1 - Input_rec%X_d2
rho_sqr = rho * rho
A_sqr = Const_A * Const_A
B_sqr = Const_B * Const_B
C_sqr = Const_C * Const_C
Beta_u_sqr = beta_u * beta_u
Beta_v_sqr = beta_v * beta_v

Q5 = Beta_u_sqr + Beta_v_sqr

gamma5 = rho_sqr / (Q5 * C_sqr)

Grad_F3_alpha_v = -Input_rec%qs * I4

end Function Grad_F3_alpha_v

real (kind=10) Function Grad_F3_beta_v(alpha_u, alpha_v, beta_u, beta_v, input_rec)
implicit none

Type Input_data
   Integer :: N_x, N_y, N_steps, NS_x, NS_y, Imid, IDebug, Ncont, Reg_w1
   Real (kind=10) :: T_step, T_initial, T_final
   Real (kind=10) :: Space_x, Space_y
   Real (kind=10) :: X_min, X_max, Y_min, Y_max
   Real (kind=10) :: Amp_1, Amp_2, Width_1, Width_2
   Real (kind=10) :: X_d1, Y_d1, Vel_1, Target_x
   Real (kind=10) :: X_d2, Y_d2, Vel_2
   Real (kind=10) :: Diff_1, Diff_2, AAAA, BBBB
   Real (kind=10) :: Nu, Theta, qs, R_phase, EPS, Om
   Real (kind=10) :: N_dp1, N_dp2, Damp_1, Damp_2, SRF
   Real (kind=10) :: Phi, Pi
   Real (kind=10) :: alpha1, alpha2, beta1, beta2, grad, inter, w1, w2, Reg_amt
End Type

Real (kind=10), intent(in) :: alpha_u, alpha_v, beta_u, beta_v
Type (Input_data), intent(in) :: input_rec
Real (kind=10) :: A_u, A_v, amp_u, amp_v, w_u, w_v, w_u_sqr, w_v_sqr, amp_u_sqr, amp_v_sqr
Real (kind=10) :: Const_A = 0.8429538023_10, Const_B = 1.177410023_10, Const_C = 1.1_10
Real (kind=10) :: I4 = 0.2954314537_10, I42 = 0.1090862907_10
Real (kind=10) :: A_sqr, B_sqr, C_sqr, Beta_u_sqr, Beta_v_sqr
Real (kind=10) :: Work_1, Work_2, Work_3, Work_4, Work_5, Work_6, Work_7, Work_8, Work_9, Work_10
Real (kind=10) :: Q1, Q2, Q3, Q4, Q5, rho, rho_sqr
Real (kind=10) :: gamma1, gamma2, gamma3, gamma4, gamma5

A_u = Input_rec%AAAA
A_v = Input_rec%BBBB
amp_u = Input_rec%Amp_1
amp_v = Input_rec%Amp_2
w_u = Input_rec%Width_1
w_v = Input_rec%Width_2
w_u_sqr = w_u * w_u
w_v_sqr = w_v * w_v
rho = Input_rec%X_d1 - Input_rec%X_d2
rho_sqr = rho * rho
A_sqr = Const_A * Const_A
B_sqr = Const_B * Const_B
C_sqr = Const_C * Const_C
Beta_u_sqr = beta_u * beta_u
Beta_v_sqr = beta_v * beta_v

Q1 = A_sqr * Beta_u_sqr + B_sqr * w_u_sqr
Q2 = A_sqr * Beta_v_sqr + B_sqr * w_u_sqr
Q3 = A_sqr * Beta_u_sqr + B_sqr * w_v_sqr
Q4 = A_sqr * Beta_v_sqr + B_sqr * w_v_sqr
Q5 = Beta_u_sqr + Beta_v_sqr

gamma1 = rho_sqr / Q1
gamma2 = rho_sqr / Q2
gamma3 = rho_sqr / Q3
gamma4 = rho_sqr / Q4
gamma5 = rho_sqr / (Q5 * C_sqr)

Work_1 = -4.0_10 * A_sqr * Beta_v * A_v * amp_v * amp_v * A_sqr * B_sqr * B_sqr * w_v_sqr * w_v_sqr / (Q4 * Q4 * Q4)
Work_2 = -2.0_10 * A_u * amp_u * amp_u * A_sqr * A_sqr * B_sqr * B_sqr * w_u_sqr * w_u_sqr / (Q2 * Q2) * exp(-gamma4)
```

```
Work_2 = Work_2 * (2.0_10 / Q2 - rho_sqr / (Q4 * Q4)) * Beta_v
Work_3 = -2.0_10 * Input_rec% qs * alpha_u * Beta_v * Beta_u_sqr * rho_sqr / (Q5 * Q5 * Q5) * exp(-gamma5)
Work_3 = Work_3 * (1.0_10 - 3.0_10 * Beta_v_sqr / Q5 + Beta_v_sqr * gamma5 / Q5)
Work_4 =  2.0_10 * A_u * amp_u * amp_u * A_sqr * A_sqr * B_sqr * Beta_v * w_u_sqr * rho_sqr / (Q2 * Q2 * Q2)
Work_4 = Work_4 * exp(-gamma2) * (1.0_10 - Beta_v_sqr / Q2 * A_sqr * (3.0_10 - rho_sqr / Q2))
Work_5 = 2.0_10 * Input_rec%qs * alpha_u * C_sqr * Beta_u_sqr * Beta_u_sqr * Beta_v / (Q5 * Q5 * Q5) * exp(-gamma5)
Work_5 = Work_5 *( 2.0_10 - gamma5)
Work_6 = -4.0_10 * Input_rec%Nu * alpha_u * Beta_u_sqr * Beta_v / (Q5 * Q5 * Q5) * exp(-gamma5)
Work_7 = (Beta_u_sqr - Beta_v_sqr - (Beta_u_sqr - 3.0_10 * Beta_v_sqr) * gamma5 - Beta_v_sqr * gamma5 * gamma5)
Work_10 = -3.0_10 / Q5 * Work_7
Work_8 = -1.0_10 + 3.0_10 * gamma5 + (Beta_u_sqr - 3.0_10 * Beta_v_sqr) * gamma5 / Q5 - gamma5 * gamma5
Work_8 = Work_8 + 2.0_10 * Beta_v_sqr * gamma5 * gamma5 / Q5
Work_9 = Work_7 * gamma5 / Q5

Grad_F3_beta_v = Work_1 + Work_2 + Work_3 + Work_4 + Work_5 + Work_6 * (Work_10 + Work_8 + Work_9)

end Function Grad_F3_beta_v

real (kind=10) Function Func_4(alpha_u, alpha_v, beta_u, beta_v, input_rec)
implicit none

Type Input_data
   Integer :: N_x, N_y, N_steps, NS_x, NS_y, Imid, IDebug, Ncont, Reg_w1
   Real (kind=10) :: T_step, T_initial, T_final
   Real (kind=10) :: Space_x, Space_y
   Real (kind=10) :: X_min, X_max, Y_min, Y_max
   Real (kind=10) :: Amp_1, Amp_2, Width_1, Width_2
   Real (kind=10) :: X_d1, Y_d1, Vel_1, Target_x
   Real (kind=10) :: X_d2, Y_d2, Vel_2
   Real (kind=10) :: Diff_1, Diff_2, AAAA, BBBB
   Real (kind=10) :: Nu, Theta, qs, R_phase, EPS, Om
   Real (kind=10) :: N_dp1, N_dp2, Damp_1, Damp_2, SRF
   Real (kind=10) :: Phi, Pi
   Real (kind=10) :: alpha1, alpha2, beta1, beta2, grad, inter, w1, w2, Reg_amt
End Type

Real (kind=10), intent(in) :: alpha_u, alpha_v, beta_u, beta_v
Type (Input_data), intent(in) :: input_rec
Real (kind=10) :: A_u, A_v, amp_u, amp_v, w_u, w_v, w_u_sqr, w_v_sqr, amp_u_sqr, amp_v_sqr
Real (kind=10) :: Const_A = 0.8429538023_10, Const_B = 1.177410023_10, Const_C = 1.1_10
Real (kind=10) :: I4 = 0.2954314537_10, I42 = 0.1090862907_10
Real (kind=10) :: A_sqr, B_sqr, C_sqr, Beta_u_sqr, Beta_v_sqr
Real (kind=10) :: Work_1, Work_2, Work_3, Work_4, Work_5, Work_6, Work_7, Work_8
Real (kind=10) :: Q1, Q2, Q3, Q4, Q5, rho, rho_sqr
Real (kind=10) :: gamma1, gamma2, gamma3, gamma4, gamma5

A_u = Input_rec%AAAA
A_v = Input_rec%BBBB
amp_u = Input_rec%Amp_1
amp_v = Input_rec%Amp_2
amp_u_sqr = amp_u * amp_u
amp_v_sqr = amp_v * amp_v
w_u = Input_rec%Width_1
w_v = Input_rec%Width_2
w_u_sqr = w_u * w_u
w_v_sqr = w_v * w_v
rho = Input_rec%X_d1 - Input_rec%X_d2
rho_sqr = rho * rho
A_sqr = Const_A * Const_A
B_sqr = Const_B * Const_B
C_sqr = Const_C * Const_C
Beta_u_sqr = beta_u * beta_u
Beta_v_sqr = beta_v * beta_v

Q1 = A_sqr * Beta_u_sqr + B_sqr * w_u_sqr
Q2 = A_sqr * Beta_v_sqr + B_sqr * w_u_sqr
Q3 = A_sqr * Beta_u_sqr + B_sqr * w_v_sqr
Q4 = A_sqr * Beta_v_sqr + B_sqr * w_v_sqr
Q5 = Beta_u_sqr + Beta_v_sqr

gamma1 = rho_sqr / Q1
gamma2 = rho_sqr / Q2
gamma3 = rho_sqr / Q3
gamma4 = rho_sqr / Q4
```

```
gamma5 = rho_sqr / (Q5 * C_sqr)

Work_1 = 2.0_10 * alpha_v * (2.0_10 * Input_rec%Nu * I42 + Input_rec%qs * Beta_v_sqr * I4)
Work_2 = - A_v * amp_v_sqr * A_sqr * B_sqr * w_v_sqr * Beta_v_sqr / Q4
Work_3 = A_u * amp_u_sqr * A_sqr * B_sqr * Beta_v_sqr * w_u_sqr / Q2 * exp(-gamma2)
Work_4 = -2.0_10 * Input_rec%Nu * alpha_u * Beta_v_sqr * Beta_u_sqr / (Q5 * Q5) * (1.0_10 - gamma5) * exp(-gamma5)
Work_5 = Input_rec%qs * alpha_u * C_sqr * Beta_v_sqr * Beta_u_sqr / Q5 * exp(-gamma5)

Func_4 = Work_1 + Work_2 + Work_3 + Work_4 + Work_5

end Function Func_4

real (kind=10) Function Grad_F4_alpha_u(alpha_u, alpha_v, beta_u, beta_v, input_rec)
implicit none

Type Input_data
   Integer :: N_x, N_y, N_steps, NS_x, NS_y, Imid, IDebug, Ncont, Reg_w1
   Real (kind=10) :: T_step, T_initial, T_final
   Real (kind=10) :: Space_x, Space_y
   Real (kind=10) :: X_min, X_max, Y_min, Y_max
   Real (kind=10) :: Amp_1, Amp_2, Width_1, Width_2
   Real (kind=10) :: X_d1, Y_d1, Vel_1, Target_x
   Real (kind=10) :: X_d2, Y_d2, Vel_2
   Real (kind=10) :: Diff_1, Diff_2, AAAA, BBBB
   Real (kind=10) :: Nu, Theta, qs, R_phase, EPS, Om
   Real (kind=10) :: N_dp1, N_dp2, Damp_1, Damp_2, SRF
   Real (kind=10) :: Phi, Pi
   Real (kind=10) :: alpha1, alpha2, beta1, beta2, grad, inter, w1, w2, Reg_amt
End Type

Real (kind=10), intent(in) :: alpha_u, alpha_v, beta_u, beta_v
Type (Input_data), intent(in) :: input_rec
Real (kind=10) :: A_u, A_v, amp_u, amp_v, w_u, w_v, w_u_sqr, w_v_sqr, amp_u_sqr, amp_v_sqr
Real (kind=10) :: Const_A = 0.8429538023_10, Const_B = 1.177410023_10, Const_C = 1.1_10
Real (kind=10) :: I4 = 0.2954314537_10, I42 = 0.1090862907_10
Real (kind=10) :: A_sqr, B_sqr, C_sqr, Beta_u_sqr, Beta_v_sqr
Real (kind=10) :: Work_1, Work_2, Work_3, Work_4, Work_5, Work_6, Work_7, Work_8
Real (kind=10) :: Q1, Q2, Q3, Q4, Q5, rho, rho_sqr
Real (kind=10) :: gamma1, gamma2, gamma3, gamma4, gamma5

A_u = Input_rec%AAAA
A_v = Input_rec%BBBB
amp_u = Input_rec%Amp_1
amp_v = Input_rec%Amp_2
amp_u_sqr = amp_u * amp_u
amp_v_sqr = amp_v * amp_v
w_u = Input_rec%Width_1
w_v = Input_rec%Width_2
w_u_sqr = w_u * w_u
w_v_sqr = w_v * w_v
rho = Input_rec%X_d1 - Input_rec%X_d2
rho_sqr = rho * rho
A_sqr = Const_A * Const_A
B_sqr = Const_B * Const_B
C_sqr = Const_C * Const_C
Beta_u_sqr = beta_u * beta_u
Beta_v_sqr = beta_v * beta_v

Q1 = A_sqr * Beta_u_sqr + B_sqr * w_u_sqr
Q2 = A_sqr * Beta_v_sqr + B_sqr * w_u_sqr
Q3 = A_sqr * Beta_u_sqr + B_sqr * w_v_sqr
Q4 = A_sqr * Beta_v_sqr + B_sqr * w_v_sqr
Q5 = Beta_u_sqr + Beta_v_sqr

gamma1 = rho_sqr / Q1
gamma2 = rho_sqr / Q2
gamma3 = rho_sqr / Q3
gamma4 = rho_sqr / Q4
gamma5 = rho_sqr / (Q5 * C_sqr)

Work_4 = -2.0_10 * Input_rec%Nu * Beta_v_sqr * Beta_u_sqr / (Q5 * Q5) * (1.0_10 - gamma5) * exp(-gamma5)
Work_5 = Input_rec%qs * C_sqr * Beta_v_sqr * Beta_u_sqr / Q5 * exp(-gamma5)

Grad_F4_alpha_u = Work_4 + Work_5
```

```
end Function Grad_F4_alpha_u

real (kind=10) Function Grad_F4_alpha_v(alpha_u, alpha_v, beta_u, beta_v, input_rec)
implicit none

Type Input_data
   Integer :: N_x, N_y, N_steps, NS_x, NS_y, Imid, IDebug, Ncont, Reg_w1
   Real (kind=10) :: T_step, T_initial, T_final
   Real (kind=10) :: Space_x, Space_y
   Real (kind=10) :: X_min, X_max, Y_min, Y_max
   Real (kind=10) :: Amp_1, Amp_2, Width_1, Width_2
   Real (kind=10) :: X_d1, Y_d1, Vel_1, Target_x
   Real (kind=10) :: X_d2, Y_d2, Vel_2
   Real (kind=10) :: Diff_1, Diff_2, AAAA, BBBB
   Real (kind=10) :: Nu, Theta, qs, R_phase, EPS, Om
   Real (kind=10) :: N_dp1, N_dp2, Damp_1, Damp_2, SRF
   Real (kind=10) :: Phi, Pi
   Real (kind=10) :: alpha1, alpha2, beta1, beta2, grad, inter, w1, w2, Reg_amt
End Type

Real (kind=10), intent(in) :: alpha_u, alpha_v, beta_u, beta_v
Type (Input_data), intent(in) :: input_rec
Real (kind=10) :: A_u, A_v, amp_u, amp_v, w_u, w_v, w_u_sqr, w_v_sqr, amp_u_sqr, amp_v_sqr
Real (kind=10) :: Const_A = 0.8429538023_10, Const_B = 1.177410023_10, Const_C = 1.1_10
Real (kind=10) :: I4 = 0.2954314537_10, I42 = 0.1090862907_10
Real (kind=10) :: A_sqr, B_sqr, C_sqr, Beta_u_sqr, Beta_v_sqr
Real (kind=10) :: Work_1, Work_2, Work_3, Work_4, Work_5, Work_6, Work_7, Work_8
Real (kind=10) :: Q1, Q2, Q3, Q4, Q5, rho, rho_sqr
Real (kind=10) :: gamma1, gamma2, gamma3, gamma4, gamma5

A_u = Input_rec%AAAA
A_v = Input_rec%BBBB
amp_u = Input_rec%Amp_1
amp_v = Input_rec%Amp_2
amp_u_sqr = amp_u * amp_u
amp_v_sqr = amp_v * amp_v
w_u = Input_rec%Width_1
w_v = Input_rec%Width_2
w_u_sqr = w_u * w_u
w_v_sqr = w_v * w_v
rho = Input_rec%X_d1 - Input_rec%X_d2
rho_sqr = rho * rho
A_sqr = Const_A * Const_A
B_sqr = Const_B * Const_B
C_sqr = Const_C * Const_C
Beta_u_sqr = beta_u * beta_u
Beta_v_sqr = beta_v * beta_v

Q1 = A_sqr * Beta_u_sqr + B_sqr * w_u_sqr
Q2 = A_sqr * Beta_v_sqr + B_sqr * w_u_sqr
Q3 = A_sqr * Beta_u_sqr + B_sqr * w_v_sqr
Q4 = A_sqr * Beta_v_sqr + B_sqr * w_v_sqr
Q5 = Beta_u_sqr + Beta_v_sqr

gamma1 = rho_sqr / Q1
gamma2 = rho_sqr / Q2
gamma3 = rho_sqr / Q3
gamma4 = rho_sqr / Q4
gamma5 = rho_sqr / (Q5 * C_sqr)

Grad_F4_alpha_v = 4.0_10 * Input_rec%Nu * I42 + 2.0_10 * Input_rec%qs * Beta_v_sqr * I4

end Function Grad_F4_alpha_v

real (kind=10) Function Grad_F4_beta_u(alpha_u, alpha_v, beta_u, beta_v, input_rec)
implicit none

Type Input_data
   Integer :: N_x, N_y, N_steps, NS_x, NS_y, Imid, IDebug, Ncont, Reg_w1
   Real (kind=10) :: T_step, T_initial, T_final
   Real (kind=10) :: Space_x, Space_y
   Real (kind=10) :: X_min, X_max, Y_min, Y_max
   Real (kind=10) :: Amp_1, Amp_2, Width_1, Width_2
```

```
   Real (kind=10) :: X_d1, Y_d1, Vel_1, Target_x
   Real (kind=10) :: X_d2, Y_d2, Vel_2
   Real (kind=10) :: Diff_1, Diff_2, AAAA, BBBB
   Real (kind=10) :: Nu, Theta, qs, R_phase, EPS, Om
   Real (kind=10) :: N_dp1, N_dp2, Damp_1, Damp_2, SRF
   Real (kind=10) :: Phi, Pi
   Real (kind=10) :: alpha1, alpha2, beta1, beta2, grad, inter, w1, w2, Reg_amt
End Type

Real (kind=10), intent(in) :: alpha_u, alpha_v, beta_u, beta_v
Type (Input_data), intent(in) :: input_rec
Real (kind=10) :: A_u, A_v, amp_u, amp_v, w_u, w_v, w_u_sqr, w_v_sqr, amp_u_sqr, amp_v_sqr
Real (kind=10) :: Const_A = 0.8429538023_10, Const_B = 1.177410023_10, Const_C = 1.1_10
Real (kind=10) :: I4 = 0.2954314537_10, I42 = 0.1090862907_10
Real (kind=10) :: A_sqr, B_sqr, C_sqr, Beta_u_sqr, Beta_v_sqr
Real (kind=10) :: Work_1, Work_2, Work_3, Work_4, Work_5, Work_6, Work_7, Work_8
Real (kind=10) :: Q1, Q2, Q3, Q4, Q5, rho, rho_sqr
Real (kind=10) :: gamma1, gamma2, gamma3, gamma4, gamma5

A_u = Input_rec%AAAA
A_v = Input_rec%BBBB
amp_u = Input_rec%Amp_1
amp_v = Input_rec%Amp_2
amp_u_sqr = amp_u * amp_u
amp_v_sqr = amp_v * amp_v
w_u = Input_rec%Width_1
w_v = Input_rec%Width_2
w_u_sqr = w_u * w_u
w_v_sqr = w_v * w_v
rho = Input_rec%X_d1 - Input_rec%X_d2
rho_sqr = rho * rho
A_sqr = Const_A * Const_A
B_sqr = Const_B * Const_B
C_sqr = Const_C * Const_C
Beta_u_sqr = beta_u * beta_u
Beta_v_sqr = beta_v * beta_v

Q1 = A_sqr * Beta_u_sqr + B_sqr * w_u_sqr
Q2 = A_sqr * Beta_v_sqr + B_sqr * w_u_sqr
Q3 = A_sqr * Beta_u_sqr + B_sqr * w_v_sqr
Q4 = A_sqr * Beta_v_sqr + B_sqr * w_v_sqr
Q5 = Beta_u_sqr + Beta_v_sqr

gamma1 = rho_sqr / Q1
gamma2 = rho_sqr / Q2
gamma3 = rho_sqr / Q3
gamma4 = rho_sqr / Q4
gamma5 = rho_sqr / (Q5 * C_sqr)

Work_3 = -4.0_10 * Input_rec%Nu * alpha_u * Beta_u * Beta_v_sqr / (Q5 * Q5) * exp(-gamma5)
Work_4 = 1.0_10 - gamma5 - Beta_u_sqr * (2.0_10 - 4.0_10 * gamma5 + gamma5 * gamma5) / Q5
Work_3 = Work_3 * Work_4

Work_5 = 2.0_10 * Input_rec%qs * alpha_u * C_sqr * Beta_v_sqr * Beta_u * exp(-gamma5) / Q5
Work_6 = 1.0_10 - Beta_u_sqr / Q5 * (1.0_10 - gamma5)
Work_5 = Work_5 * Work_6

Grad_F4_beta_u = Work_3 + work_5

end Function Grad_F4_beta_u

real (kind=10) Function Grad_F4_beta_v(alpha_u, alpha_v, beta_u, beta_v, input_rec)
implicit none

Type Input_data
   Integer :: N_x, N_y, N_steps, NS_x, NS_y, Imid, IDebug, Ncont, Reg_w1
   Real (kind=10) :: T_step, T_initial, T_final
   Real (kind=10) :: Space_x, Space_y
   Real (kind=10) :: X_min, X_max, Y_min, Y_max
   Real (kind=10) :: Amp_1, Amp_2, Width_1, Width_2
   Real (kind=10) :: X_d1, Y_d1, Vel_1, Target_x
   Real (kind=10) :: X_d2, Y_d2, Vel_2
   Real (kind=10) :: Diff_1, Diff_2, AAAA, BBBB
   Real (kind=10) :: Nu, Theta, qs, R_phase, EPS, Om
```

```
   Real (kind=10) :: N_dp1, N_dp2, Damp_1, Damp_2, SRF
   Real (kind=10) :: Phi, Pi
   Real (kind=10) :: alpha1, alpha2, beta1, beta2, grad, inter, w1, w2, Reg_amt
End Type

Real (kind=10), intent(in) :: alpha_u, alpha_v, beta_u, beta_v
Type (Input_data), intent(in) :: input_rec
Real (kind=10) :: A_u, A_v, amp_u, amp_v, w_u, w_v, w_u_sqr, w_v_sqr, amp_u_sqr, amp_v_sqr
Real (kind=10) :: Const_A = 0.8429538023_10, Const_B = 1.177410023_10, Const_C = 1.1_10
Real (kind=10) :: I4 = 0.2954314537_10, I42 = 0.1090862907_10
Real (kind=10) :: A_sqr, B_sqr, C_sqr, Beta_u_sqr, Beta_v_sqr
Real (kind=10) :: Work_1, Work_2, Work_3, Work_4, Work_5, Work_6, Work_7, Work_8
Real (kind=10) :: Q1, Q2, Q3, Q4, Q5, rho, rho_sqr
Real (kind=10) :: gamma1, gamma2, gamma3, gamma4, gamma5

A_u = Input_rec%AAAA
A_v = Input_rec%BBBB
amp_u = Input_rec%Amp_1
amp_v = Input_rec%Amp_2
amp_u_sqr = amp_u * amp_u
amp_v_sqr = amp_v * amp_v
w_u = Input_rec%Width_1
w_v = Input_rec%Width_2
w_u_sqr = w_u * w_u
w_v_sqr = w_v * w_v
rho = Input_rec%X_d1 - Input_rec%X_d2
rho_sqr = rho * rho
A_sqr = Const_A * Const_A
B_sqr = Const_B * Const_B
C_sqr = Const_C * Const_C
Beta_u_sqr = beta_u * beta_u
Beta_v_sqr = beta_v * beta_v

Q1 = A_sqr * Beta_u_sqr + B_sqr * w_u_sqr
Q2 = A_sqr * Beta_v_sqr + B_sqr * w_u_sqr
Q3 = A_sqr * Beta_u_sqr + B_sqr * w_v_sqr
Q4 = A_sqr * Beta_v_sqr + B_sqr * w_v_sqr
Q5 = Beta_u_sqr + Beta_v_sqr

gamma1 = rho_sqr / Q1
gamma2 = rho_sqr / Q2
gamma3 = rho_sqr / Q3
gamma4 = rho_sqr / Q4
gamma5 = rho_sqr / (Q5 * C_sqr)

Work_1 = -2.0_10 * A_v * amp_v_sqr * A_sqr * B_sqr * w_v_sqr * Beta_v / Q4 * (1.0_10 - A_sqr * Beta_v_sqr / Q4)
Work_1 = Work_1 + 4.0_10 * alpha_v * Input_rec%qs * Beta_v * I4

Work_2 = 2.0_10 * A_u * amp_u_sqr * A_sqr * B_sqr * w_u_sqr * Beta_v / Q2 * exp(-gamma2)
Work_3 = 1.0_10 - Beta_v_sqr * A_sqr / Q2 + A_sqr * Beta_v_sqr * rho_sqr / (Q2 * Q2)
Work_2 = Work_2 * Work_3

Work_4 = -4.0_10 * Input_rec%Nu * alpha_u * Beta_v * Beta_u_sqr / (Q5 * Q5) * exp(-gamma5)
Work_5 = 1.0_10 - gamma5 - 2.0_10 * Beta_v_sqr * (1.0_10 - gamma5) / Q5 + Beta_v_sqr * gamma5 / Q5
Work_5 = Work_5 + Beta_v_sqr * (1.0_10 - gamma5) * gamma5 / Q5
Work_4 = Work_4 * Work_5

Work_6 = 2.0_10 * Input_rec%qs * alpha_u * Beta_u_sqr * Beta_v * exp(-gamma5) * C_sqr / Q5
Work_7 = 1.0_10 - Beta_v_sqr / Q5 + Beta_v_sqr * gamma5 / Q5
Work_6 = Work_6 * Work_7

Grad_F4_beta_v = Work_1 + Work_2 + Work_4 + Work_6

end Function Grad_F4_beta_v
```

B

Appendix A

Published Work

In addition to presenting the research at a conference, the author has also co-authored the following paper

- Smyth, N.F., and Tope, B., 2016, "Beam on beam control: beyond the particle approximation," Journal of Nonlinear Optical Physics and Materials,**25**, 1650046.

Bibliography

[1] J. Scott Russell, "Report on waves", *Fourteenth meeting of the British Association for the Advancement of Science,* Vol. 311, (1844).

[2] Lord Rayleigh, "On waves". *Philosophical Magazine,* Series 5. 1 (4): 257–279, (1876).

[3] J. Boussinesq,"Théorie de l'intumescence liquide, applele onde solitaire ou de translation, se propageant dans un canal rectangulaire". *Comptes Rendus de l'Academie des Sciences,* 72: 755–759, (1871).

[4] H. Lamb, *Hydrodynamics,* Cambridge University Press, 6th Edition, (1933).

[5] N. J. Zabusky, M. D. Kruskal, "Interactions of "solitons" in a collision less plasma and the recurrence of initial states", *Phys. Rev. Lett.,* **15**, no. 6, p 240, (1965).

[6] C. S. Gardner, J. M. Greene, M. D. Kruskal, R. M. Miura "Method for Solving the Korteweg-de-Vries Equation", *Phys. Rev. Lett.,* **19**, no. 19, pp. 1095–1097, (1967).

[7] V. L. Ginzburg, L. D. Landau,*Sov Phys JETP* **20**, 1064, (1950)

[8] V. L. Ginzburg, *Sov Phys JETP*, **2**, 589, (1956)

[9] V. L. Ginzburg, L. P. Pitaevskii, *Sov Phys JETP* **7**, 858, (1958)

[10] R. Y. Chiao, E. Garmire, C. H. Towner, *Phys. Rev. Lett.,* **15**, 479, (1964)

[11] V. I. Talanov, *Radiophysics*, **7**, 254, (1964)

[12] D. J. Benney, A. C. Newell, *J Math Phys*, **46**, 133, (1967)

[13] H. S. Eisenberg, Y. Silberberg, R. Morandotti, A. R. Boyd, J. S. Aitchison, *Phys. Rev. Lett.,* **81**, 3383, (1998)

[14] N. K. Efremidis, D. N. Christodoulides, J. W. Fleischer, O. Cohen, M. Segev, *Phys. Rev. Lett.,* **91**, 213906, (2003)

[15] C. J. Pethick, H. Smith H, "Bose- Einstein Condensation in Dilute Gases", *Cambridge University Press*, (2002)

[16] A. C. Scott, *Encyclopedia of Nonlinear Science*, Routledge, Taylor and Francis Group, New York, NY, (2005).

[17] D. Duncan, *Heriot-Watt University, Dept. of Mathematics, Soliton web page,* **http://www.ma.hw.ac.uk/solitons/**, image is at URL http://www.ma.hw.ac.uk/solitons/soliton1.html

[18] C. Finot and K. Hammani, *Université de Bourgogne, Dijon, Bourgogne, FRANCE, Department of Physics of the University of Bourgogne quipe Solitons, Lasers et Communications Optiques* image is at URL https://commons.wikimedia.org/wiki/File:Soliton_hydro.jpg

[19] J. Schmaltz, *MODIS Land Rapid Response Team, NASA GSFC*, image is at URL https://www.nasa.gov/image-feature/oceanic-nonlinear-internal-solitary-waves-from-the-lombok-strait

[20] Kebes, *Wikipedia*, https://en.wikipedia.org/wiki/Soliton

[21] H. Afshari, B. Olyaeefar and H. Khoshsima *The Refractive Index Grating Formation in Azo DyeDoped Nematic Liquid Crystal*, Molecular Crystals and Liquid Crystals, **561**, 1:36, (2012), image is at URL https://www.researchgate.net/publication/227854864_The_Refractive_Index_Grating_Formation_in_Azo_DyeDoped_Nematic_Liquid_Crystal/figures?lo=1

[22] T. Dauxois, M. Peyrard, *Physics of Solitons*, Cambridge University Press, Cambridge, UK, (2006).

[23] M. A. Porter, N. J. Zabusky, B. Hu, D. K. Campbell, *Fermi, Pasta, Ulam and the Birth of Experimental Mathematics*, American Scientist 97(6): 214222. doi:10.1511/2009.78.214. , (2009b).

[24] M. Remoissenet, *Waves Called Solitons: Concepts and Experiments*, 3rd edition, Springer-Verlag, Berlin, Germany, (1999).

[25] B. A. Malomed, *Soliton Management in Periodic Systems*, Springer-Verlag, Berlin, Germany, (2005).

[26] M. Centurion, M. A. Porter, P. G. Kevrekidis, D. Psaltis, *Nonlinearity Management in Optics: Experiment, Theory, and Simulation*, *Phys. Rev. Lett.,* **97(3)**, : 033903. doi:10.1103/physrevlett.97.033903. , (2006).

[27] P. G. Kevrekidis, D. J. Frantzeskakis, R. Carretero-Gonzlez, *Emergent Phenomena in BoseEinstein Condensates: Theory and Experiment*, Springer-Verlag, Berlin, Germany, (2008).

[28] R. Carretero-Gonzlez, D. J. Frantzeskakis,P. G. Kevrekidis, *Nonlinear Waves in BoseEinstein Condensates: Physical Relevance and Mathematical Techniques.* Nonlinearity 21: R139R202. doi:10.1088/0951-7715/21/7/r01. , (2008).

[29] V. E. Zakharov, *Stability of Periodic Waves of Finite Amplitude on the Surface of a Deep Fluid.* Zhurnal Prikladnoi Mekhaniki i Tekhniki Fizika 9: 190194. doi:10.1007/bf00913182. [Journal of Applied Mechanics and Technical Physics 9: 190194 (1968)].

[30] A. S. Davydov, *Biology and Quantum Mechanics*, Pergamon Press, Oxford, UK, (1982).

[31] P. Engels, C. Atherton, M. A. Hoefer, *Observation of Faraday Waves in a BoseEinstein Condensate*, Physical Review Letters 98: 095301. doi:10.1103/physrevlett.98.095301. , (2007).

[32] G. P. Agrawal, *Nonlinear Fibre Optics,* (Academic, New York, 1989).

[33] A. Hasegawa, *Optical Solitons in Fibres,* "nd ed. (Springer, Berlin, 1990).

[34] A. Alberucci and G. Assanto, 2007, *J. Opt. Soc. Am. B,* **24**, 2314.

[35] Assanto, G., Peccianti, M. and Conti, C., 2003, *Optics & Photonic News*, **14**, 45–48.

[36] G. Assanto, A. Piccardi, A. Alberucci, S. Residori and U. Bortolozzo, 2009, *Photon. Lett. Poland,* **1(4)**, 151–153.

[37] C. García-Reimbert, A.A. Minzoni, T.R. Marchant, N.F. Smyth, and A.L. Worthy, "Dipole soliton formation in a nematic liquid crystal in the nonlocal limit," *Physica D, 237* **240**, 1088–1102 (2008).

[38] C. García-Reimbert, A.A. Minzoni, N.F. Smyth, and A.L. Worthy, "Large-amplitude nematicon propagation in a liquid crystal with local response," *J. Opt. Soc. Am. B,* **23**, 2551–2558 (2006).

[39] B. A. Malomed, "Variational methods in nonlinear fiber optics and related fields," *Progress in Optics 43*, E. Wolf, Elsevier Science, 71–193 (2002).

[40] D. Anderson,"Variational approach to nonlinear pulse propagation in optical fibres," *Phys. Rev. A 27,* 3135–3145 (1983).

[41] Skuse, B.D., and Smyth, N.F., 2009, *Phys. Rev. A,* **79**, 063806.

[42] M. and G. Assanto, "Nematicons," *Phys. Rep.,* **516**, 147–208 (2012).

[43] J. L. Lagrange, *Mécanique Analytique,* Cambridge University Press, (2009).

[44] G. Assanto, *Nematicons, spatial optical solitons in nematic liquid crystals,* John Wiley and Sons, New York (2012).

[45] G.B. Whitham, *Linear and Nonlinear Waves,* J. Wiley and Sons, New York (1974).

[46] M. Peccianti, A. de Rossi, G. Assanto, A. De Luca, C. Umeton and I.C. Khoo, "Electrically assisted self-confinement and waveguiding in planar nematic liquid crystal cells," *Appl. Phys. Lett.,* **77**, 7–9 (2000).

[47] Y.V. Izdebskaya, J. Rebling, A.S. Desyatnikov and Y.S. Kivshar, "Observation of vector solitons with hidden vorticity," *Opt. Lett.,* **37**, 767–769 (2012).

[48] A. S. Davydov, "Solitons and energy transfer along protein molecules." *Journal of Theoretical Biology,* **66**(2): 379387, (1977).

[49] C. Conti, M. Peccianti and G. Assanto, "Observation of optical spatial solitons in a highly nonlocal medium," *Phys. Rev. Lett.,* **91**, 073901 (2003).

[50] C. Conti, M. Peccianti and G. Assanto, "Route to nonlocality and observation of accessible solitons," *Phys. Rev. Lett.,* **92**, 113902 (2004).

[51] M. Peccianti, K.A. Brzdąkiewicz and G. Assanto, "Nonlocal spatial soliton interactions in nematic liquid crystals," *Opt. Lett.,* **27**, 1460–1462 (2002).

[52] G. Assanto and M. Peccianti, "Spatial solitons in nematic liquid crystals," *IEEE J. Quantum Electron.,* **39**, 13–21 (2003).

[53] G. Assanto, M. Peccianti, K.A. Brzdąiewicz, A. de Luca and C. Umeton, "Nonlinear wave propagation and spatial solitons in nematic liquid crystals," *J. Nonlinear Opt. Phys. Mater.,* **12**, 123–134 (2003).

[54] W. Hu, T. Zhang, Q. Guo, L. Xuan and S. Lan, "Nonlocality-controlled interaction of spatial solitons in nematic liquid crystals," *Appl. Phys. Lett.,* **89**, 071111 (2006).

[55] Y.S. Kivshar and G.P. Agrawal, *Optical Solitons. From Fibers to Photonic Crystals,* Academic Press, San Diego (2003).

[56] A. Fratalocchi, A. Piccardi, M. Peccianti and G. Assanto, Nonlinear management of the angular momentum of soliton clusters: Theory and experiments," *Phys. Rev. Lett.,* **75**, 063835 (2007).

[57] A. Fratalocchi, M. Peccianti, C. Conti and G. Assanto, "Spiraling and cyclic dynamics of nematicons," *Mol. Cryst. Liq. Cryst.,* **421** 197–207 (2004).

[58] G. Assanto, N.F. Smyth and A.L. Worthy, "Two colour, nonlocal vector solitary waves with angular momentum in nematic liquid crystals," *Phys. Rev. A,* **78**, 013832 (2008).

[59] G. Assanto, C. García-Reimbert, A.A. Minzoni, N.F. Smyth, and A.L. Worthy, "Lagrange solution for three wavelength solitary wave clusters in nematic liquid crystals," *Physica D,* **240**, 1213–1219 (2011).

[60] J.F. Henninot, M. Debailleul, M. Warenghem, "Tunable non-locality of thermal non-linearity in dye doped nematic liquid crystal," *Mol. Cryst. Liq. Cryst.,* **375**, 631–640 (2002).

[61] J.F. Henninot, J.F. Blach and M. Warenghem, "Experimental study of the nonlocality of spatial optical solitons excited in nematic liquid crystal," *J. Opt. A: Pure Appl. Opt.,* **9**, 20–25 (2007).

[62] Y.V. Izdebskaya, V.G. Shvedov, A.S. Desyatnikov, W.Z. Krolikowski, Milivoj Belić, G. Assanto and Y.S. Kivshar, "Counterpropagating nematicons in bias-free liquid crystals," *Opt. Express,* **18**, 3258–3263 (2010).

[63] M. Peccianti, C. Conti, G. Assanto, A. De Luca and C. Umeton, "Routing of anisotropic spatial solitons and modulational instability in liquid crystals," *Nature,* **432**, 733–737 (2004).

[64] I.C. Khoo, *Liquid Crystals: Physical Properties and Nonlinear Optical Phenomena,* Wiley, New York (1995).

[65] R.H. Enns, *It's a Nonlinear World,* Springer, New York (2010).

[66] S.V. Antipov, M.V. Nezlin, E.N. Snezhkin, A.S. Trubnikov, *Rossby utosoliton and stationary model of the Jovian Great Red Spot,* Nature 323:238, 1986.

[67] A.P. Ingersoll, *Jupiter's Great Red Spot. A free atmospheric vortex,* Science, 182:1346 (1995).

[68] S.V. Serak, N.V. Tabiryan, M. Peccianti and G. Assanto, "Spatial soliton all-optical logic gates," *IEEE Photon. Tech. Lett.,* **18**, 1287–1289 (2006).

[69] A. Piccardi, A. Alberucci, U. Bortolozzo, S. Residori and G. Assanto, "Soliton gating and switching in liquid crystal light valve," *Appl. Phys. Lett.,* **96**, 071104 (2010).

[70] A. Piccardi, A. Alberucci, N. Tabiryan and G. Assanto, "Dark nematicons," *Opt. Lett.,* **36**, 1356–1358 (2011).

[71] M. A. Karpierz,"Solitary waves in liquid crystalline waveguides," *Phys. Rev. E*, **66**, 0336603 (2002).

[72] M. Peccianti, C. Conti, G. Assanto, A. De Luca and C. Umeton, "All-optical switching and logic gating with spatial solitons in liquid crystals," *Appl. Phys. Lett.,* **81**, 3335–3337 (2002).

[73] A.A. Minzoni, N.F. Smyth and A.L. Worthy, "Modulation solutions for nematicon propagation in non-local liquid crystals," *J. Opt. Soc. Amer. B,* **24**, 1549–1556 (2007).

[74] J.M.L. MacNeil, N.F. Smyth and G. Assanto, "Exact and approximate solutions for solitary waves in nematic liquid crystals," *Physica D,* **284**, 1–15 (2014).

[75] G. Assanto, J.M.L. MacNeil and N.F. Smyth, "Diffraction-induced instability of coupled dark solitary waves," *Optics Letter,* **40**, 8 (2015).

[76] B.D. Skuse and N.F. Smyth, "Two-colour vector soliton interactions in nematic liquid crystals in the local response regime," *Phys. Rev. A*, **77**, 013817 (2008).

[77] B.D. Skuse and N.F. Smyth, "Interaction of two colour solitary waves in a liquid crystal in the nonlocal regime," *Phys. Rev. A,* **79**, 063806 (2009).

[78] D.J. Kaup and A.C. Newell, "Solitons as particles, oscillators, and in slowly changing media: a singular perturbation theory," *Proc. Roy. Soc. Lond. A 361,* 413–446 (1978).

[79] A. Alberucci, M. Peccianti, G. Assanto, A. Dyadyusha and M. Kaczmarek, "Two-color vector solitons in nonlocal media," *Phys. Rev. Lett. 97,* 153903 (2006).

[80] G. Assanto, A.A. Minzoni, M. Peccianti and N.F. Smyth, "Optical solitary waves escaping a wide trapping potential in nematic liquid crystals: modulation theory," *Phys. Rev. A,* **79**, 033837 (2009).

[81] Y. Izdebskaya, W. Krolikowski, N.F. Smyth and G. Assanto, "Vortex stabilization by means of spatial solitons in nonlocal media," *J. Opt.,* **18**, 054006 (2016).

[82] A.B. Aceves, J.V. Moloney and A.C. Newell, "Theory of light-beam propagation at nonlinear interfaces. I Equivalent-particle theory for a single interface," *Phys. Rev. A,* **39**, 1809–1827 (1989).

[83] E.A. Kuznetsov and A.M. Rubenchik, "Soliton stabilization in plasmas and hydrodynamics," *Phys. Reports*, **142**, 103–165 (1986).

[84] F.W. Dabby and J.R. Whinnery, "Thermal self-focusing of laser beams in lead glasses," *Appl. Phys. Lett.,* **13**, 284–286 (1968).

[85] C. Rotschild, M. Segev, Z. Xu, Y.V. Kartashov and L. Torner, "Two-dimensional multipole solitons in nonlocal nonlinear media," *Opt. Lett.,* **31**, 3312–3314 (2006).

[86] C. Rotschild, B. Alfassi, O. Cohen and M. Segev, "Long-range interactions between optical solitons," *Nature Phys.,* **2**, 769–774 (2006).

[87] M. Segev, B. Crosignani, A. Yariv and B. Fischer, "Spatial solitons in photorefractive media," *Phys. Rev. Lett.,* **68**, 923–926 (1992).

[88] A. Cheskidov, D.D. Holm, E. Olson and E.S. Titi, "On a Leray-α model of turbulence," *Proc. R. Soc. Lond. A*, **461**, 629–649 (2005).

[89] A. Ilyin, E.M. Lunasin and E.S. Titi, "A modified-Leray-α subgrid scale model of turbulence," *Nonlinearity*, **19**, 879–897 (2006).

[90] B. Malomed, "Variational methods in nonlinear fiber optics and related fields," *Prog. Opt.*, **43**, 71–193 (2002).

[91] W.L. Kath and N.F. Smyth, "Soliton evolution and radiation loss for the nonlinear Schrodinger equation," *Phys. Rev. E*, **51**, 1484–1492 (1995).

[92] G. Assanto, N.F. Smyth and W. Xia, "Modulation analysis of nonlinear beam refraction at an interface in liquid crystals," *Phys. Rev. A*, **84**, 033818 (2011).

[93] G. Assanto, N.F. Smyth and W. Xia, "Refraction of nonlinear light beams in nematic liquid crystals," *J. Nonlin. Opt. Phys. Mater.*, **21**, 1250033 (2012).

[94] A.A. Minzoni, N.F. Smyth, A.L. Worthy and Y.S. Kivshar, "Stabilization of vortex solitons in nonlocal nonlinear media," *Phys. Rev. A*, **76**, 063803 (2007).

[95] Y.S. Kivshar and X. Yang, "Perturbation-induced dynamics of dark solitons," *Phys. Rev. E*, **49**, 1657–1670 (1994).

[96] Y.S. Kivshar and W. Krolikowski, "Lagrangian approach for dark solitons," *Opt. Comm.*, **114**, 353–362 (1995).

[97] G. Theocharis, P. Schmelcher, M.K. Oberthaler, P.G. Kevrekidis and D.J. Frantzeskakis, "Lagrangian approach to the dynamics of dark matter-wave solitons," *Phys. Rev. A*, **72**, 023609 (2005).

[98] Q. Kong, Q. Wang, O. Bang and W. Krolikowski, "Analytical theory of dark nonlocal solitons," *Opt. Lett.*, **35**, 2152–2154 (2010).

[99] L. Chen, Q. Wang, M. Shen, H. Zhao, Y.-Y. Lin, C.-C. Jeng, R.-K. Lee and W. Krolikowski, "Nonlocal dark solitons under competing cubicquintic nonlinearities," *Opt. Lett.*, **38**, 13–15 (2013).

[100] Y.V. Kartashov and L. Torner, "Gray spatial solitons in nonlocal nonlinear media," *Opt. Lett.*, **32**, 946–948 (2007).

[101] S. Pu, C. Hou, K. Zhan and C. Yuan, "Dark and gray solitons in nematic liquid crystals," *Physica Scripta*, **85**, 015402 (2012).

[102] S. Pu, C. Hou, K. Zhan, C. Yuan and Y. Du, "Lagrangian approach for dark soliton in nonlocal nonlinear media," *Opt. Commun.*, **285**, 3631–3635 (2012).

[103] Y. Lin and R.-K. Lee, "Dark-bright soliton pairs in nonlocal nonlinear media," *Opt. Express.*, **15**, 8781–8786 (2007).

[104] W. Chen, Q. Kong, M. Shen, Q. Wang and J. Shi, "Polarized vector dark solitons in nonlocal Kerr-type self-defocusing media," *Phys. Rev. A*, **87**, 013809 (2013).

[105] A.A. Minzoni, N.F. Smyth and Z. Xu, "Stability of an optical vortex in a circular nematic cell," *Phys. Rev. A*, **81**, 033816 (2010).

[106] N.F. Smyth and W. Xia, "Refraction and instability of optical vortices at an interface in a liquid crystal," *J. Phys. B: At. Mol. Opt. Phys.*, **45**, 165403 (2012).

[107] G. Assanto, A.A. Minzoni, N.F. Smyth and A.L. Worthy, "Refraction of nonlinear beams by localised refractive index changes in nematic liquid crystals," *Phys. Rev. A*, **82**, 053843 (2010).

[108] G. Assanto, B.D. Skuse and N.F. Smyth, "Solitary wave propagation and steering through light-induced refractive potentials," *Phys. Rev. A,* **81**, 063811 (2010).

[109] G. Assanto, A.A. Minzoni, N.F. Smyth and A.L. Worthy, "Refraction of nonlinear beams by localised refractive index changes in nematic liquid crystals," *Phys. Rev. A,* **82**, 053843 (2010).

[110] A. Alberucci, G. Assanto, A.A. Minzoni and N.F. Smyth, "Scattering of reorientational optical solitary waves at dielectric perturbations," *Phys. Rev. A,* **85**, 013804 (2012).

[111] G. Assanto, A.A. Minzoni and N.F. Smyth, "Vortex confinement and bending with nonlocal solitons," *Opt. Lett.,* **39**, 509–512 (2014).

[112] G. Assanto, A.A. Minzoni and N.F. Smyth, "Deflection of nematicon-vortex vector solitons in liquid crystals," *Phys. Rev. A,* **89**, 013827 (2014).

[113] M. Abramowitz and I.A. Stegun, *Handbook of Mathematical Functions with Formulas, Graphs and Mathematical Tables,* Dover Publications, Inc., New York (1972).

[114] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery, *Numerical Recipes in Fortran. The Art of Scientific Computing,* Cambridge University Press, (1992).

[115] B. Fornberg and G.B. Whitham, "A numerical and theoretical study of certain nonlinear wave phenomena," *Phil. Trans. Roy. Soc. London A 289*, 373–403 (1978).

[116] G. Assanto, T. R. Marchant, A.A. Minzoni and N.F. Smyth, "Reorientational versus Kerr dark and gray solitary waves using modulation theory," *Phys. Rev. E,* **84**, 066602 (2011).

[117] R. Burden and J. Faires, *Numerical Analysis,* Brookes/Cole, Boston, Massachusetts (2011).

[118] R. Resnick, D. Halliday, *Physics,* Wiley International Edition, John Wiley and Sons, (1966).

[119] H. Goldstein, *Classical Mechanics,* Addison-Wesley, (1980).

[120] W. Kaplan, *Advanced Calculus,* 2nd edition, Addison-Wesley, (1973).

[121] L.N. Trefethen, *Spectral Methods in MATLAB*, SIAM, Philadephia (2000).

[122] T.F. Chan and T. Kerkhoven, "Fourier methods with extended stability intervals for KdV," *SIAM J. Numer. Anal.,* **22**, 441–454 (1985).

[123] A. Piccardi, G. Assanto, L. Lucchetti, and F. Simoni "All-optical steering of soliton waveguides in dye-doped liquid crystals," *Appl. Phys. Lett.,* **93**, 171104 (2008).

[124] A. Piccardi, A. Alberucci, and G. Assanto, "Self-turning self-confined light beams in guest-host media," *Phys. Rev. Lett.,* **104**, 213904 (2010).

[125] Benjamin, T.B., 1967, "Instability of periodic wavetrains in nonlinear dispersive systems," *Proceedings of the Royal Society of London Series A, Mathematical and Physical Sciences,* **299**, 59–75.

[126] Benjamin, T.B., and Feir, J.E., 1967, "The disintegration of wave trains on deep water. Part 1. Theory," *Journal of Fluid Mechanics,* **27**, 417–430.

[127] A.C. Newell, *Solitons in Mathematics and Physics,* SIAM, Philadelphia (1985)

[128] M. Matuszewski, W. Krolikowski and Y.S. Kivshar, "Soliton interactions and transformations in colloidal media," *Phys. Rev. A* **79**, 023814 (2009).

[129] M. Matuszewski, W. Krolikowski and Y.S. Kivshar, "Bistable solitons in colloidal media," *Photon. Lett. Poland* **1**, 4–6 (2009).

[130] G. Assanto, A. Minzoni, and N. F. Smyth, "Light self-localization in nematic liquid crystals: modelling solitons in reorientational media," *J. Nonl. Opt. Phys. Mat.,* **18**, 657–691 (2009).